TKScope®

嵌入式智能仿真开发平台



User Guide

用户指南

产品制造商与技术支持

广州致远电子有限公司

地址：广州市天河区车陂路黄洲工业区 3 栋 2 楼

邮编：510660

电话：（020）22644372　28872453（销售服务）

（020）22644360　28267808（技术支持）

传真：（020）38601859

E-mail：TKS@zlgmcu.com（技术支持）

网址：www.embedtools.com

销售与服务网络

广州周立功单片机发展有限公司

地址：广州市天河区天河北路 689 号光大银行大厦 12 楼 F4 室

邮编：510630

电话：（020）38730916　38730917　38730976　38730977

传真：（020）38730925

网址：www.zlgmcu.com

广州专卖店

地址：广州市天河区新赛格电子城 203～204 室

电话：（020）87578634　87569917　87578842

传真：（020）87578842

深圳周立功

地址：深圳市深南中路 2070 号电子科技大厦 C 座 4 楼 D 室

电话：（0755）83781768　83781788　83782922

传真：（0755）83793285

成都周立功

地址：成都市一环路南二段 1 号数码同人港 401 室

（磨子桥立交西北角）

电话：（028）85439836　85432683　85437446

传真：（028）85437896

重庆周立功

地址：重庆市石桥铺科园一路二号大西洋国际大厦 1611 室

（赛格电子市场）

电话：（023）68796438　68796439

传真：（023）68796439

武汉周立功

地址：武汉市洪山区广埠屯珞瑜路 158 号 12128 室

（华中电脑数码市场）

电话：（027）87168497　87168297　87168397

传真：（027）87163755

北京周立功

地址：北京市海淀区知春路 113 号银网中心 A 座 1207～1208 室

（中发电子市场斜对面）

电话：（010）62536178　62536179　82628073

传真：（010）82614433

上海周立功

地址：上海市北京东路 668 号科技京城东座 7E 室

电话：（021）53083452　53083453　53083496

传真：（021）53083491

杭州周立功

地址：杭州市天目山路 217 号江南电子大厦 502 室

电话：（0571）28139611　28139612　28139613

传真：（0571）28139621

南京周立功

地址：南京市珠江路 280 号珠江大厦 2006 室

电话：（025）83613221　83603005　83603500

传真：（025）83613271

西安办事处

地址：西安市长安北路 54 号太平洋大厦 1201 室

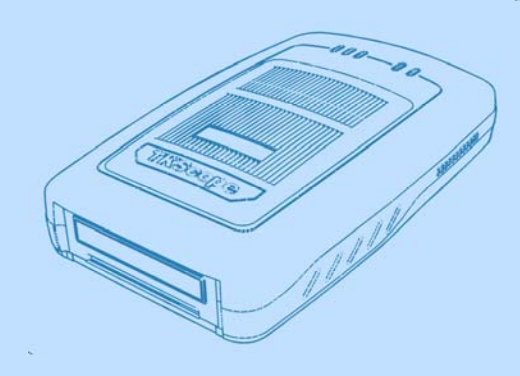
电话：（029）87881296　83063000　87881295

传真：（029）87880865

目录

第 1 章

认识 TKScope 仿真器



1.1 功能特点和背景资料

TKScope 嵌入式智能仿真开发平台是广州致远电子有限公司 2008 年隆重推出上市的一款高性能通用型综合仿真开发平台，支持仿真全系列的 **8051、ARM、DSP、AVR、C166、C251、MX** 等内核；与当前全部主流 IDE 环境无缝嵌接，如 **TKStudio、Keil、ADS、IAR、CCS、RealView、AVRStudio** 等，保证您的开发平台始终如一，并具备其高级调试功能。同时，TKScope 内嵌 **64 路**专业的**逻辑分析仪**，zlgLogic 高级软件全面支持。



与全部主流 IDE 环境无缝嵌接，TKStudio，CCS，Keil，ADS，RealView，IAR，AVRStudio 等。

- 独创的新概念仿真框架，囊括超过 3000 多种主流芯片，无一遗漏。
- 高速 USB 联机通讯，2500K 字节/秒的用户代码下载速度，快如闪电。
- 真正专业的内置逻辑分析仪，64 路/200M 速度/1M 深度，明察秋毫。
- 全球首创的全资源 On the fly，真正运行中无插入操作，方便自如。
- 全球首创 Super Trace / Super View / Super Break 特色调试，得心应手。
- 真正的 IAP/ISP 功能支持，涵盖 Flash 数据操作和代码运行，细致入微。
- 内部 1M 字节程序和数据空间，并支持代码数据分组调试，前所未有。

本文主要讲解 TKScope 仿真 8051 内核的性能以及在主流 IDE 环境下的使用方法。



TKScope 仿真器能够支持 8051 内核仿真的具体型号有如下几种：

- K 系列：K3 / K5 / K8 / K9。

TKScope 仿真器使用当今超大规模集成电路的最新技术以及规范的硬件模块化设计，全面提升 TKScope 仿真器在嵌入式系统开发中的表现能力。TKScope 仿真器主要在以下的嵌入式领域为用户提供仿真、测试等开发手段：

- MCU 微处理器的仿真和辅助开发。
- ARM 处理器的仿真和辅助开发。
- DSP 数字信号处理器的仿真和辅助开发。
- FPGA/CPLD 等 EDA 技术的开发。
- 逻辑信号和模拟信号的测试和开发。
- 其它专用领域的信号测试和技术开发。





TKScope 仿真器将带给用户一个全新的开发理念和感受，让用户在体验其杰出性能的同时也满足了视觉上的享受。

TKScope 仿真器经过全方面的专业设计优化，不仅进军的仿真测试领域广泛，而且其性能也是非常杰出优异的，是以往仿真器所不能同日而语的，在应用中能够达到让您叹为观止的性能。

- 与主流芯片厂商合作，保证 TKScope 仿真器使用最佳的仿真技术，以达到最佳的仿真性能。
- 与主流 IDE 厂商合作，保证 TKScope 仿真器能在众多主流 IDE 环境中无缝嵌接。
- 采用标准物理接口和硬件可重构技术，全新概念的柔性仿真框架，从容应对当今和未来的 MCU 仿真测试。
- 采用 QuickUSB 技术，保证最大 2500KB /S 的通讯和下载速度。
- 采用 QuickBUS 高速总线技术，稳定仿真超过 80MHz 的处理器。
- 采用 Auto Volt 技术，仿真电压 1.8V-5.5V，满足全电压范围的芯片仿真。
- 长达 600 小时运行计数器，支持时间标签(10ns 分辨率)。
- 内置 PLL 频率发生器，可以自动产生用户设置的 20K-100MHz 运行时钟，时间精度为 0.001%。

QuickUSB	最佳仿真技术	逻辑分析仪
QuickBUS	主流 IDE 环境	硬件自检
外部信号 组合触发	512KB 性能分析器	512KB Super Trace
仿 真 电 压 1.8V-5.5V	支持数千种 芯片仿真	512KB 代码 执行覆盖分析
512KB 加彩 运行轨迹显示	内外时钟检测 20K-120MHz	精密运行计时器 10ns/600 小时
最大 512KB 内/外部存储器	512KB Von Neumann	入出口线 100%保护
支持 6/12Clock 静态/动态切换	时间断点 Super Break	On the fly 运行中操作
支持 ALE 静态/动态关闭	Remap 功能 1Byte 精度	Bank 分组支持

- 支持用户目标板的时钟输入和仿真头上的外部振荡晶体，并进行时钟有效性检查。
- 512KB 全地址范围内的代码执行覆盖分析，加彩运行轨迹显示。
- 512KB 全地址范围内的代码读取覆盖显示/数据读取覆盖显示/数据写覆盖显示。
- 512KB 代码和数据的性能分析，并以统计图形的方式表现代码和数据的执行效率。
- 512KB 超大容量实时 Super Trace 超级跟踪功能，32 位同步时间记录，全球首个 8051 特殊功能寄存器 SFR 跟踪记录。
- 内建 Bank 选项，支持最大 512KB 分组程序开发。内部最大采用 6×512KB 仿真空间，支持 8×64KB 的 Bank 仿真模式。
- 内部代码空间/数据空间均支持 1Byte 精度的 ReMap 功能，方便用户仿真不同资源的芯片。
- 支持运行仿真器内部/外部 0-64KB 的 ROM 和 RAM。
- 支持 64KB 全范围的冯诺曼 Von Neumann 结构。
- 采用 On the fly 仿真技术，其透明特性，允许用户在运行中查看/修改全部资源。
- 多种复杂断点设计，特设时间断点功能，精确度到 ns，方便用户特殊仿真要求。
- 采用高速信号跟踪技术，真正支持 ALE 静态关闭或动态关闭，不限制对 ALE 信号的非常规切换。
- 自动感知 6/12Clock 时钟，并支持动态切换和静态切换，不限制用户对时钟信号的非常规切换。
- 所有仿真器入出口线 100%保护，避免使用中误操作引起仿真器的损坏，保护用户投资。
- 带有硬件自检功能，方便用户判断仿真器工作异常的原因，解决了用户在使用仿真器过程中，出现莫明其妙的问题时束手无策的尴尬局面。

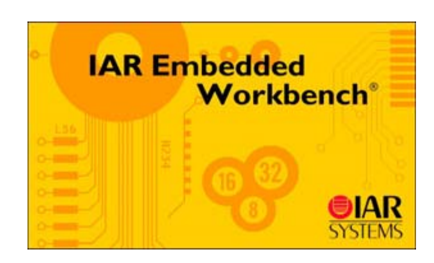
1.2 技术优势

与主流 IDE 厂商合作，保证 TKScope 与全部主流 IDE 无缝嵌接，保证你的开发平台始终统一，并具备其高级调试功能。

- **TKStudio**，致远公司，中/英文界面，多内核编译/调试环境，强大内置编辑器。
- **ZlgLogic**，致远公司，中/英文界面，逻辑分析仪测试环境，强大内置测试功能。
- **CCS**，TI 公司，英文界面，DSP 编译调试环境。
- **Keil**，Keil 公司，英文界面，8051/251/C166/ARM 编译/调试环境。
- **ADS**，ARM 公司，英文界面，全 ARM 内核编译/调试环境。
- **RealView**，ARM 公司，英文界面，全 ARM 内核编译/调试环境。
- **IAR**，IAR 公司，英文界面，多内核编译/调试环境。
- **AVRStudio**，ATMEL 公司，英文界面，AVR 编译/调试环境。













最新的仿真技术支持——支持 8051 / ARM / DSP / AVR / C166 / MX 内核。

- 与各大 IC 厂商紧密合作，保证每款芯片使用最合理的仿真技术。
- 支持仿真全系列 8051/ARM/DSP/AVR/C166/MX 等内核，囊括超过 3000 多种主流芯片。
- 主流 IC 厂商支持。

Acer Labs	Aeroflex UTMC	AMD
Analog Device	ATMEL	Cirrus Logic
CML Microcircuts	Cybernetic Micro	Dallas
Freescale	Honeywell	Hynix
Infineon	Intel	ISSI
Luminary Micro	Marvell	Megawin
OKI	NXP	Samsung
SHARP	SST	ST
STC	SyncMOS	TI
WinBOND	更多未列入厂商	

全资源 On the fly 操作，全球首例最新专利技术

- 内部全部寄存器运行中可见。
- 内部数据区域运行中可见。
- 外部数据区域运行中可见。
- 5×512KB 断点运行中任意设置和取消。
- 数据性能分析运行中结果可见。
- 代码覆盖和数据覆盖运行中结果可见。
- 程序运行加彩轨迹运行中可见。
- 程序指针与源代码运行中紧密关联。

Super Trace 超级跟踪，全球首例最新专利技术

- 全球首个 8051 全范围 SFR 跟踪记录，更好的协助用户分析程序运行轨迹。
- 最大 512KB 超大容量实时 Super Trace 超级跟踪功能。
- 跟踪记录 ACC/B/DPTR/SP 等全部 SFR。
- 跟踪记录内部 4 组 R0-R7 寄存器。
- 同时记录 48-bit 同步时间记录标签（Time Stamp），10ns 精度。
- 同时记录程序地址指针 PC。

8051 代码/数据分组调试仿真支持

- 突破 8051 的 64KB 代码数据限制，可仿真最大 8 分组 64KB 空间。
- 支持代码分组（Bank）调试，最大 8×64KB。
- 支持数据分组（Bank）调试，最大 8×64KB。
- 支持 4 路仿真器外部 Bank 控制信号输入。
- 支持无限制数量 MCU 内部 Bank 控制信号。

真正的 IAP/ISP 功能仿真支持

- 突破传统仿真技术缺陷，真正实现内部 Flash 的 IAP/ISP 的仿真。
- 用户程序代码可自动/快速下载到 MCU 内部/外部 Flash 中，并支持用户自定义 Flash 编程算法。
- 支持 Flash 代码用途的仿真，包括代码运行的代码读取。
- 支持 Flash 数据用途的仿真，包括读取和写入操作。
- 支持 Flash 本身操作的仿真，包括扇区擦除、全片擦除、字节写等 Flash 操作。
- ARM 内核中无限制个数的 Flash 断点。

同步信号的输入/输出

- 1×路仿真监控信号输出，可同步触发外部测试仪器，如示波器和逻辑分析仪等。
- 1×路逻辑分析仪触发信号输出，可同步触发外部测试仪器，如示波器和逻辑分析仪等。
- 2×路外部同步信号输入，可进行逻辑编程，触发内部逻辑分析仪和停止仿真器模块运行。

内嵌多种专业分析工具，协助改善程序结构优化代码性能

- 代码覆盖分析，最大 512KB 容量，运行中结果可见。
- 轨迹加彩显示，最大 512KB 容量，运行中结果可见。
- 数据覆盖分析，最大 512KB 容量，运行中结果可见。
- 代码性能分析，最大 512KB 容量，运行中结果可见。
- 数据性能分析，最大 512KB 容量，运行中结果可见。
- 代码运行跟踪，最大 512KB 容量，48-bit 位同步时间记录和 PC 记录。
- 超级代码跟踪，最大 512KB 容量，48-bit 位同步时间记录和 PC 记录，SFR 和 R0-R7 记录。
- 复杂断点设计，包括数据读写操作断点，地址范围判断断点，堆栈溢出断点等。
- 时间断点功能，最大 600 小时/10ns 精度，方便用户特殊仿真要求。

真正专业的内嵌逻辑分析仪，zlglogic 全面支持

- 最大 64 路输入信号，200M 采样速度，1MB 存储深度。
- 创新的触发测量方式和崭新的分析测量手段，令测量更加简单快捷。
- 可设置边缘/电平/总线等基本触发方式和高级复杂触发方式，方便易用。
- 人性化软件轻松完成信号测量、触发设置、动态帮助、软件升级等功能。
- 内部 32 路缺省仿真 MCU 信号，参与显示和触发。
- 仿真模块和逻辑分析仪模块紧密关联，可同时独立运行或停止。
- 多文档结构可让您在测量的同时观察和比较其它数据。
- 强大的数据导出功能支持对测量信号进行二次分析成为可能。
- 柔性频率设置突破传统的 1、2、5 进制，使得测量更加精确。
- 动态升级的硬件算法使您的测量手段与时并进。

1.3 型号分类和基本性能

目前，广州致远电子有限公司推出的 TKScope K 系列仿真器的型号和基本性能见下表，其中带有（*）标记的为 TKScope 仿真器独有或杰出特性。

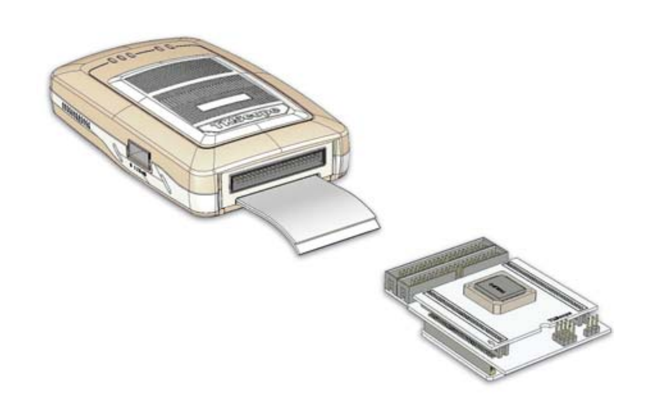
仿真器型号	K3	K5	K8	K9
基本性能	8051 通用型	8051 通用型	8/16/32 位通用型	8/16/32 位通用型
仿真类型	8051	8051	8051，ARM，AVR，C166，C251，MX…	8051，ARM，DSP，AVR，C166，C251，MX…
通讯方式*	USB 1.1，500KB/S			USB2.0，2500KB/S
仿真频率*	80MHz 仿真频率，20K-120MHz 内部时钟			
逻辑分析板		内置	内置	内置
仿真存储器	6×128KB	6×128KB	6×256KB	6×512KB
影子存储器	128KB	128KB	256KB	512KB
Bank 分组支持*				8×64KB 分组支持
仿真电压*	1.8-5.5V 程控步进			
Super View*				
运行中内核观察*	√	√	√	√
运行中 PC 观察*	√	√	√	√
运行中时间观察*	√	√	√	√
运行中覆盖观察*	√	√	√	√
运行中性能观察*		√	√	√
Super Access*				
运行中代码操作*	√	√	√	√
运行中数据操作*	√	√	√	√
运行中断点操作*	√	√	√	√
加彩程序轨迹*	√	√	√	√
精密运行计时器*	10ns 精度，600 小时记录			
基本断点	5×128KB	5×128KB	5×256KB	5×512KB
时间断点	64bits 时间断点	64bits 时间断点	64bits 时间断点	64bits 时间断点
SuperBreak*	√	√	√	√
外部信号组合触发*	√	√	√	√
专业逻辑分析仪*		64 路/128KB/200MHz	64 路/512KB/200MHz	64 路/1MB/200MHz
程序时效分析		128KB 范围时效分析	256KB 范围时效分析	512KB 范围时效分析
数据时效分析		128KB 时效分析	256KB 时效分析	512KB 时效分析
代码覆盖分析		128KB 代码覆盖分析	256KB 代码覆盖分析	512KB 代码覆盖分析
程序运行跟踪		128KB 帧运行跟踪	256KB 帧运行跟踪	512KB 帧运行跟踪
超级结果跟踪*		128KB 帧运行跟踪	256KB 帧运行跟踪	512KB 帧运行跟踪
IDE 支持*	TKStudio，Keil，IAR，ADS，RealView，zlglogic，AVRStudio，CCS 等			
升级		√	√	√

注：带*为 TKScope 独创或突出性能。

1.4 硬件构成和仿真头种类

一台完整的 TKScope 仿真器主要由主机和仿真头两部分构成，如果说主机是仿真器的灵魂，那么仿真头就是仿真器必不可少的左膀右臂，两者缺一不可。

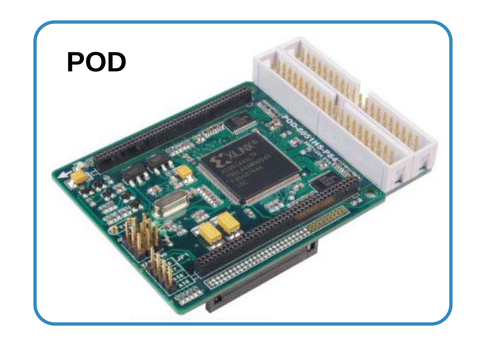




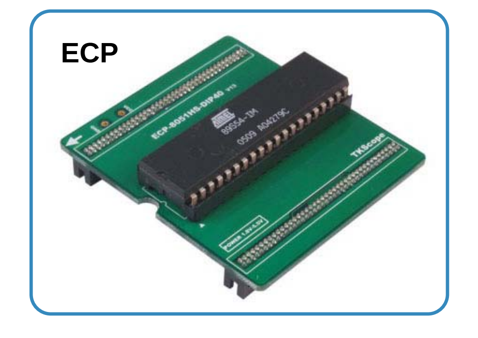
仿真头种类

TKScope 仿真器的仿真头由三部分组合构成，采用“POD 主板＋ECP 板＋ADP 板”结构。用户需要根据实际仿真要求，选择正确的 POD 主板、ECP 板和 ADP 板，组合成合适的仿真头。

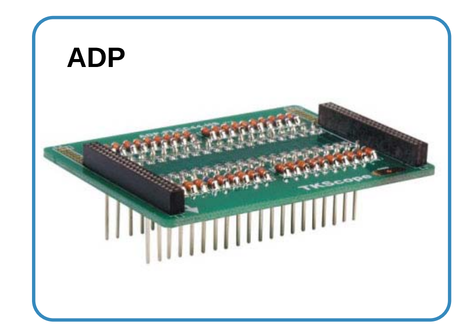
POD



ECP



ADP





请您在使用之前确认仿真要求，以便选择合适的仿真头组件。



请您参照下面的仿真头组件类型列表，根据说明信息选择您所需要的组件。仿真头选择不当或错误，可能会导致仿真失败。

POD 主板类型列表

POD 型号	说明
POD-8051HS-P84	可仿真不同厂商/各种封装的 8051 芯片
POD-8051BS-P84	可仿真不同厂商/各种封装的 Bondout 芯片，如 LPC700 和 LPC900 系列
POD-JTAG-ARM-DP20	可仿真不同厂商的全部 ARM 芯片，包括 ARM7/ARM9/ARM11/XSCALE/Cortex 系列
POD- XDS560B-P14	可仿真 TI 全部 JTAG 接口的 DSP 芯片（高速 560 模式，适用于 DK 系列）
POD-XDS510P-P14	可仿真 TI 全部 JTAG 接口的 DSP 芯片（中速 510 模式，适用于 K 系列）
POD-JTAG-AVR-P10	可仿真全部 JTAG/DebugWire 方式的 AVR 芯片
POD-OneWire-ADI-P2	可仿真全部的 ADI 增强型 51 内核芯片
POD-OCDS-DP16	可仿真全部 OCDS 接口的芯片，如 XC800/XC166/XE167/XC2000 系列
POD-JC2-C8051F-DP10	可仿真 Silicon Labs 全部的 C8051F 芯片
其它或陆续推出的高性能 POD 组件	

ECP 板类型列表

ECP 型号		说明
通用型号	ECP-8051HS-PLCC44	所有标准 PLCC44 封装的 80C51 系列，完全插入式仿真
	ECP-8051HS-DIP40	所有标准 DIP40 封装的 80C51 系列，完全插入式仿真
	ECP-8051OCDS-P42	所有 80C51 系列的 OCDS 在线仿真，42 脚标准
专用型号	ECP-P87C591HS-PLCC44	P8xC591 专用，PLCC44 封装
	ECP-P8xC552HS-PLCC68	P8xC55x 专用，PLCC68 封装
	ECP-AT89C51CC03HS-PLCC44	AT89C51CC01_03 专用，PLCC44 封装
	ECP-AT80C51SND1CHS-PLCC84-QFP80	AT8xC51SND1C、AT89C5132 专用，PLCC84、QPF80 封装
	ECP-8051HS-PLCC44-QFP44-C505	Infineon C505 专用，QFP44 封装
	ECP-SABC517HS-PLCC84	Infineon SAB80C517/C517 专用，PLCC84 封装
	ECP-W79E648HS-PLCC68	W79E648/W79E548 专用，PLCC68 封装
其它或陆续推出的高性能 ECP 组件		

ADP 板类型列表

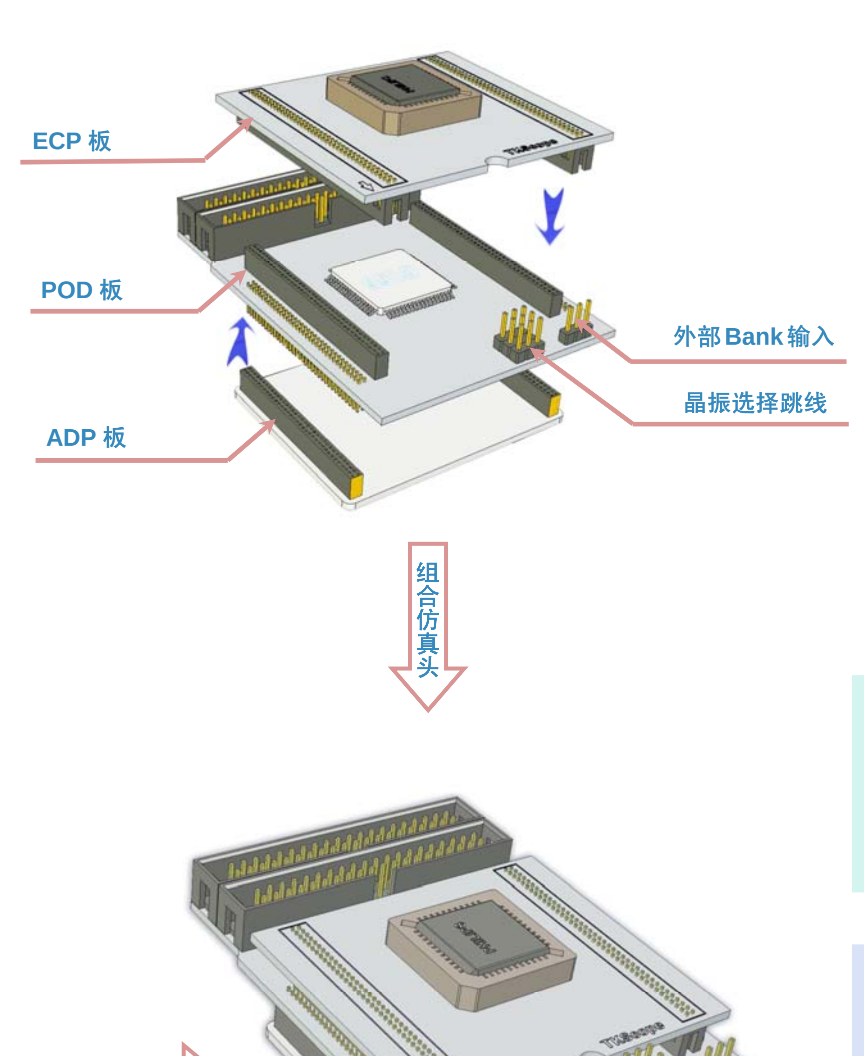
ADP 型号	说明
ADP-8051HS-PLCC84	所有标准 PLCC84 封装的 80C51 系列，PLCC84 接口
ADP-8051HS-PLCC68	所有标准 PLCC68 封装的 80C51 系列，PLCC68 接口
ADP-8051HS-PLCC52	所有标准 PLCC52 封装的 80C51 系列，PLCC52 接口
ADP-8051HS-PLCC44	所有 PLCC44 封装的 80C51 系列，PLCC44 接口
ADP-8051HS-DIP40	所有 DIP40 封装的 80C51 系列，DIP40 接口
ADP-8051OCDS-IDC42	所有 80C51 系列的 OCDS 在线仿真，42 脚 IDC 接口
其它或陆续推出的高性能 ADP 组件	

仿真头组合

用户根据仿真芯片的信息选择正确的 POD 主板、ECP 板和 ADP 板，然后组合成仿真头。实际要仿真的芯片放到 ECP 板上的芯片插座内，然后把 ECP 板插到 POD 主板上，最后再把 ADP 板插到 POD 板的下面，这样就组合成了一个完整的仿真头。



注意！用户在组合过程中，按照板上的箭头方向提示进行安装，要保持三块板子的方向一致性，否则可能会导致仿真器不能正常工作甚至损坏。



POD 主板是仿真头的核心部分。板上有用于晶振选择的跳线以及外部 Bank 输入控制端。板上 JP1 处的插针用于外部 Bank 输入控制，共有 8 种地址形式，用户根据实际仿真需要进行控制。晶振选择的跳线，出厂时的状态是选择仿真头上的晶振，连接 ONBD 侧；用户也可以选择目标板上的晶振，连接 TARG 侧即可。

POD 主板**上面**连接着 **ECP 板**。ECP 板上的底座用于安装用户实际仿真的芯片。

POD 主板**下面**连接着 **ADP 板**。ADP 板用于插到用户目标板的仿真接口。

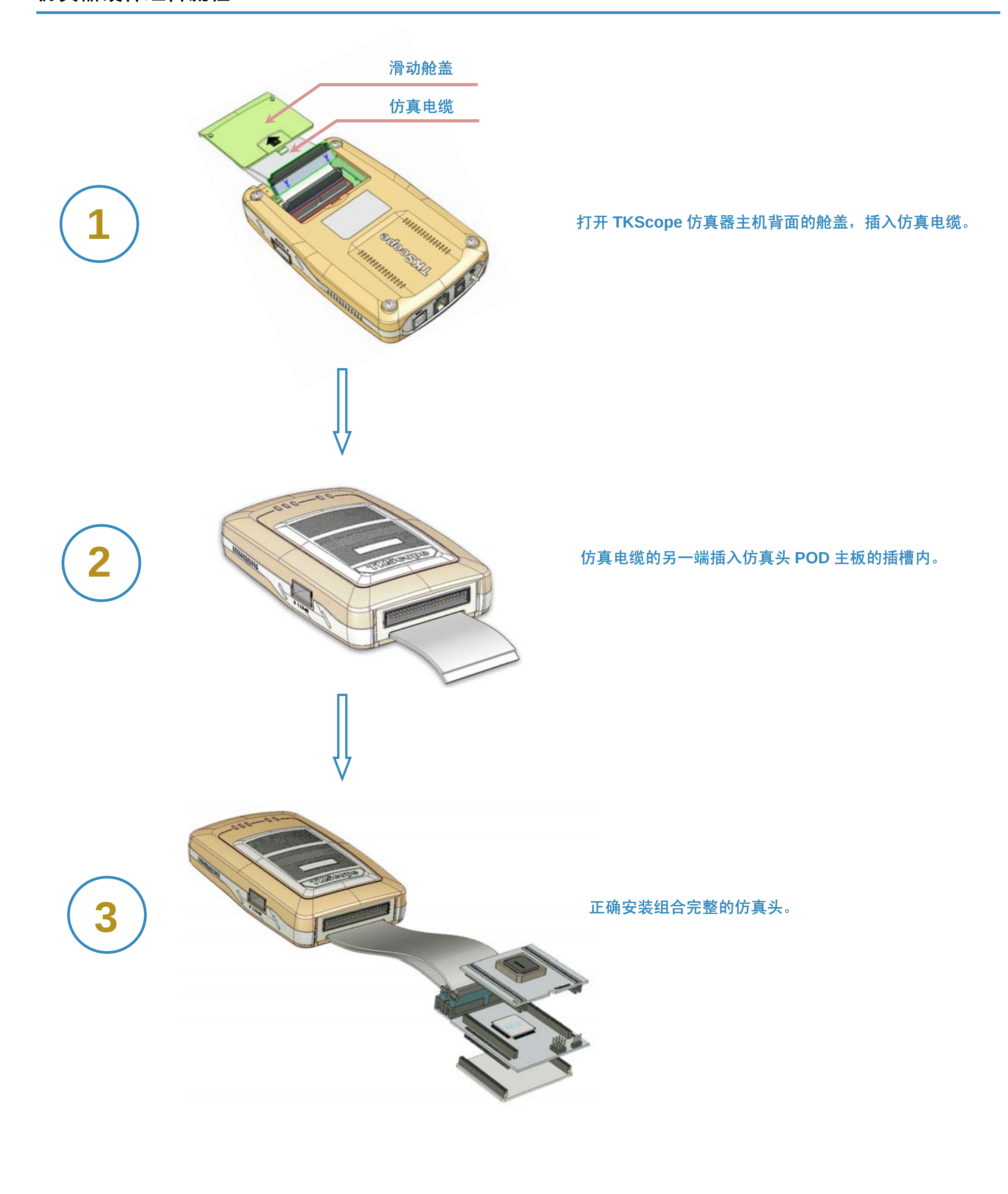
注意！ECP 板上安装的芯片必须是用户实际要仿真的芯片，该芯片无需用户烧写程序。但是，切记该芯片一定不能处于加密状态，否则仿真无法进行。



举例：用户仿真 P89V51RD2FA 芯片，此芯片是 PLCC44 封装，用户目标板上的仿真接口预留是 DIP40 形式的。

此时，POD 主板要选择 POD-8051HS 型号，ECP 板要选择 ECP-8051HS-PLCC44 型号，ADP 板要选择 ADP-8051HS-DIP40 型号。

仿真器硬件组合流程



1 打开 TKScope 仿真器主机背面的舱盖，插入仿真电缆。

2 仿真电缆的另一端插入仿真头 POD 主板的插槽内。

3 正确安装组合完整的仿真头。

第 2 章

使用 TKScope 仿真器



2.1 驱动安装方法

TKScope 仿真器支持多种开发/调试平台，其中 TKStudio/Keil 中英文开发平台使用较普及，本章主要讲解 TKScope 仿真器在 Keil IDE 开发环境下的入门操作。



开始之前！

- TKScope 仿真器在 Keil IDE 环境下使用，需要正确的安装 Keil 软件和 TKScope 仿真器的驱动。
- Keil 软件的安装过程很简单，按照提示信息进行操作即可，在此不做详细描述。
- TKScope 仿真器的驱动安装过程和方法，本节将详细讲解。

仿真器驱动安装过程

1 双击 TKScopeSetup_C51.EXE，系统会弹出如图 2.1 所示的对话框。

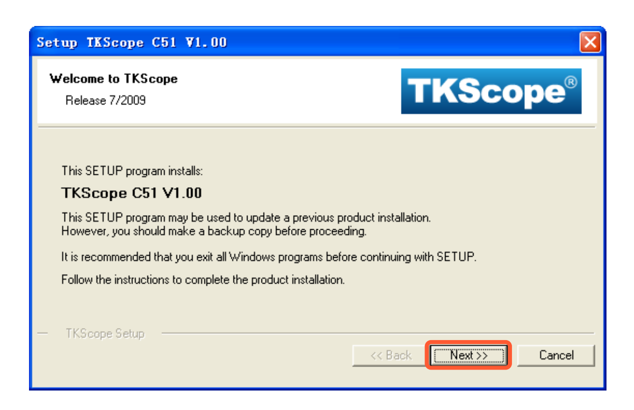


图 2.1　安装驱动提示框

2 点击图 2.1 中【**Next**】选项，系统会弹出如图 2.2 所示的对话框。选中【**I agree to…**】，然后点击【**Next**】进行下一步操作。

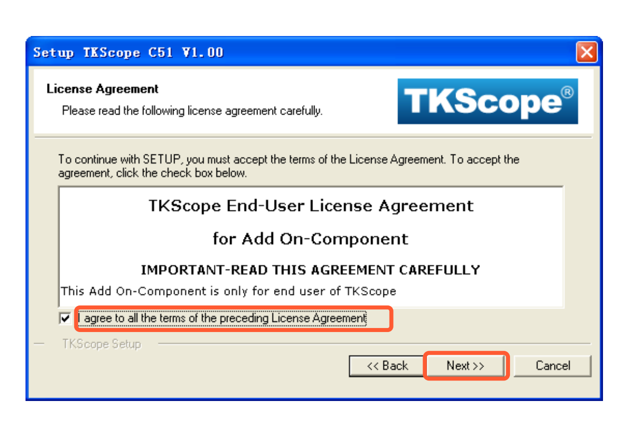


图 2.2　协议许可提示框

3 此时，需要指定仿真器驱动程序的安装路径，如图 2.3 所示。注意：驱动程序必须安装在 Keil 软件的安装路径下！

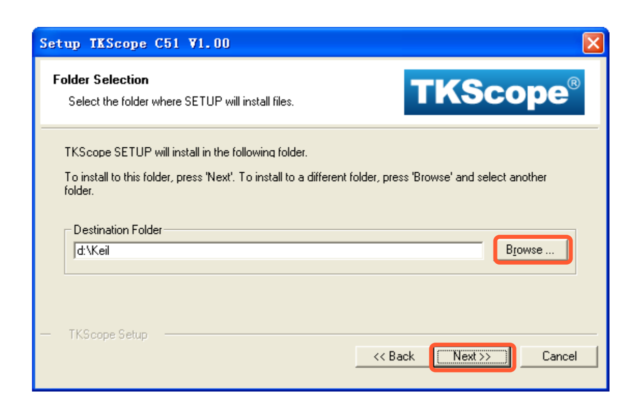


图 2.3　指定驱动安装路径

4 点击图 2.3 中【**Next**】选项，系统会弹出如图 2.4 所示的对话框，用户根据提示填写自己的信息。

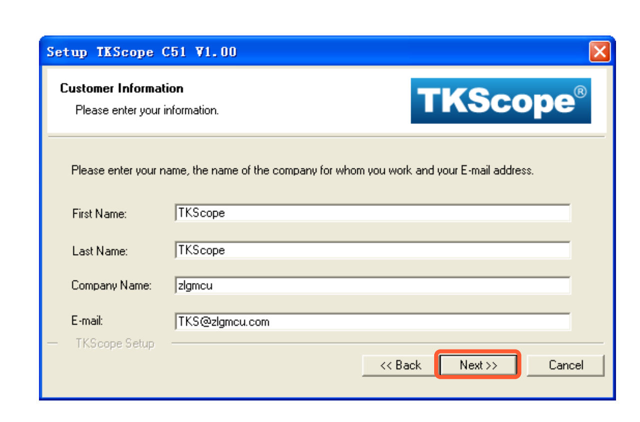


图 2.4　填写用户信息

5 点击图 2.4 中【Next】选项，即可开始仿真器驱动程序的安装，安装过程如图 2.5 所示。

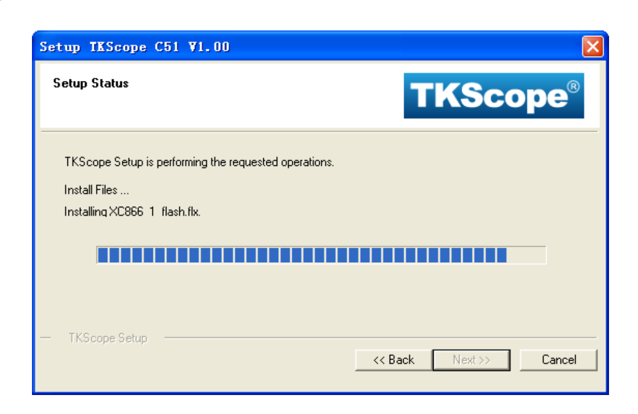


图 2.5 驱动安装进度提示框

6 仿真器驱动程序正确成功的安装完毕的结果，如图 2.6 所示，用户点击【Finish】即可结束。

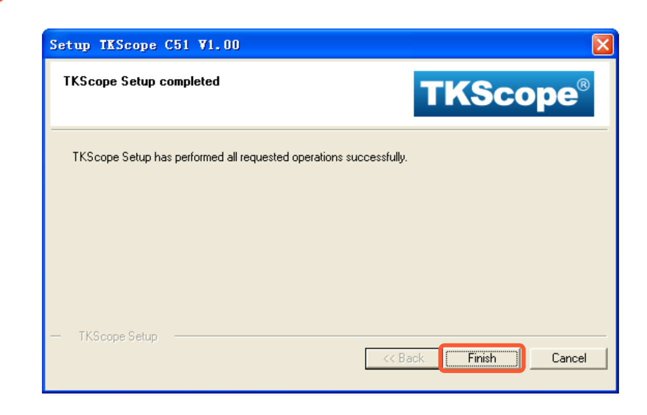


图 2.6 驱动成功安装完成

仿真器驱动安装结果

仿真器驱动正确成功安装之后，可以在仿真环境设置界面中，查看仿真器驱动安装的结果。

选择【Project】菜单下【Options for Target】选项，进入仿真环境设置界面。

在【Debug】选项里，硬件仿真驱动选择下拉菜单中，可以看到 TKScope 仿真器的驱动选项，如图 2.7 所示。

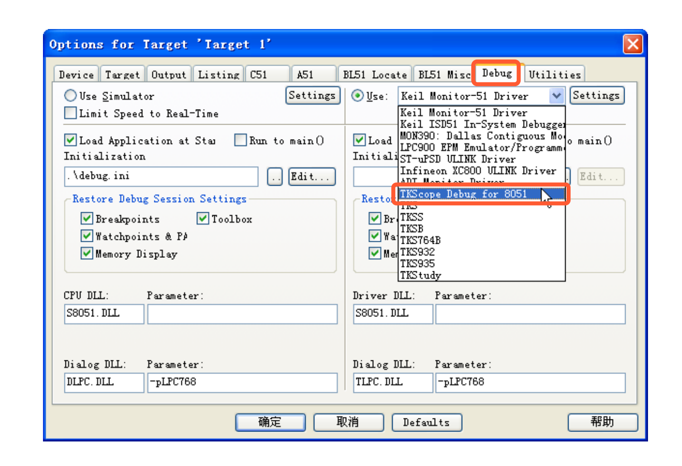


图 2.7 查看仿真器驱动安装结果

USB 设备驱动安装过程

1 驱动程序安装之后，第一次使用仿真器，给其上电时，一般情况下系统会弹出如图 2.8 所示的对话框。此时，需要指定 USB 设备驱动的具体位置。

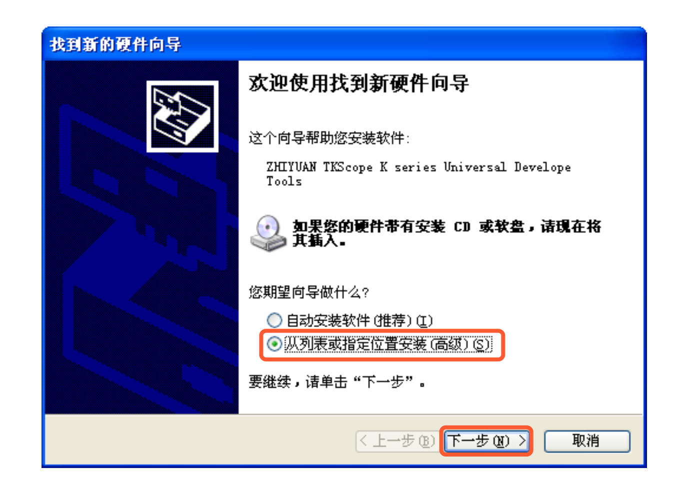


图 2.8 找到新硬件向导

2 选择图 2.8 中的【**从列表或指定位置安装（高级）**】选项，然后点击【**下一步**】，此时系统会弹出如图 2.9 所示的对话框。

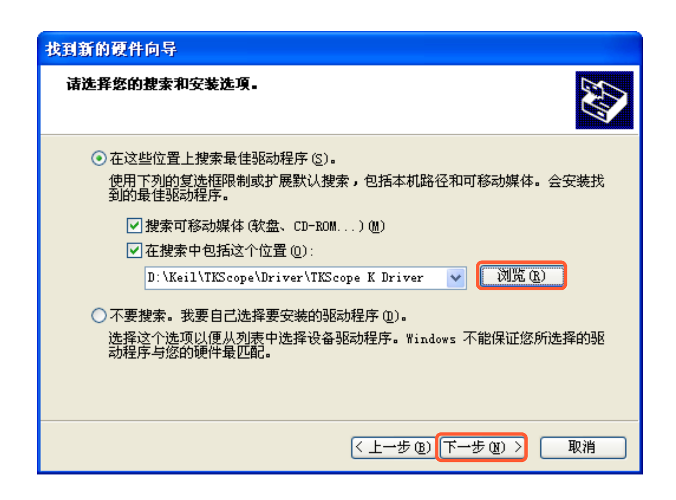


图 2.9　选择驱动提示框

3 点击图 2.9 中的【**浏览**】选项，进入如图 2.10 所示的界面。按照 TKScope 仿真器驱动安装的路径，找到驱动文件 **TKScope K Driver**，然后点击【**确定**】。

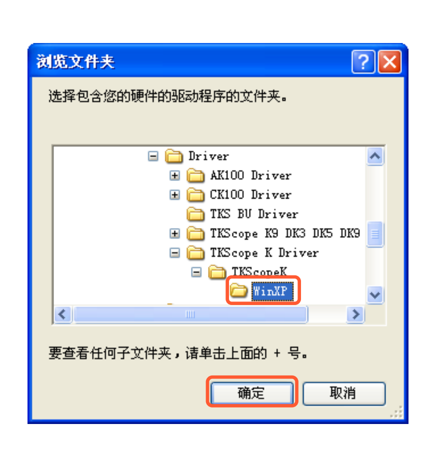


图 2.10　指定驱动具体位置

4 指定正确的驱动之后，系统自动进行安装，安装过程中会弹出如图 2.11 所示的对话框。此时需要选择【**仍然继续**】选项，让安装过程继续进行。



图 2.11　硬件安装过程

5 驱动安装完毕，系统会弹出如图 2.11 所示的对话框，提示用户已经完成驱动的安装。此时，点击【**完成**】即可，至此，驱动程序安装完毕。

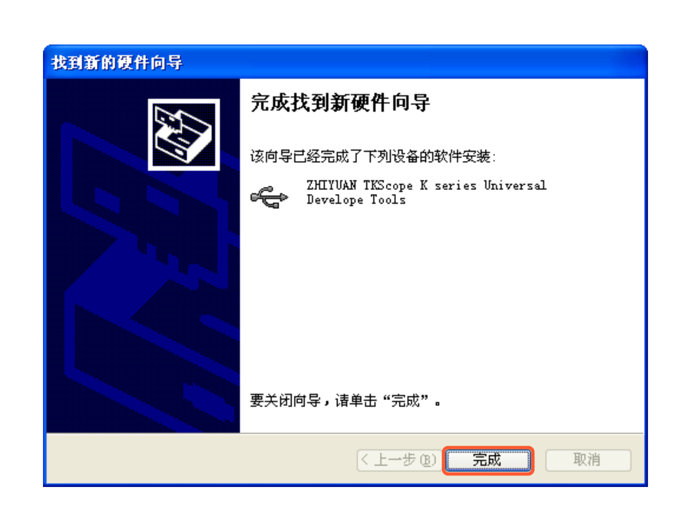


图 2.12　新硬件安装完成

1

系统正确安装驱动后，可以通过查看设备管理器看到当前的硬件设备。使用鼠标右键点击【**我的电脑**】，选择【**属性**】，进入如图 2.13 所示的界面。

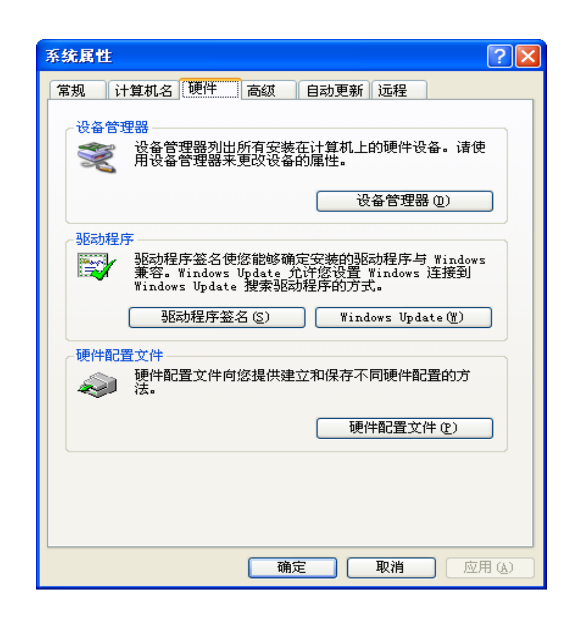


图 2.13 系统属性

2

在图 2.13 中，点击【**设备管理器**】，进入如图 2.14 所示的界面。此时，可以在【**通用串行总线控制器**】一栏内看到系统识别到的新安装的硬件设备。

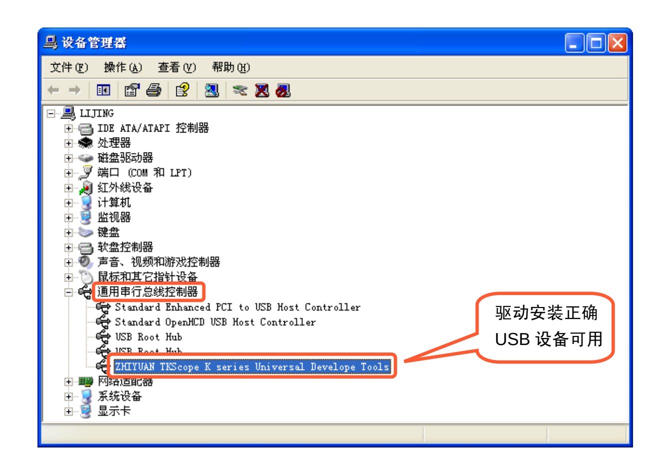


图 2.14 正确的安装新硬件结果

3

如果系统没有安装新硬件的驱动或驱动安装不正确，那么会看到如图 2.15 所示的界面。系统检测到新的 USB 设备，但是没有找到正确的驱动，无法正常使用。

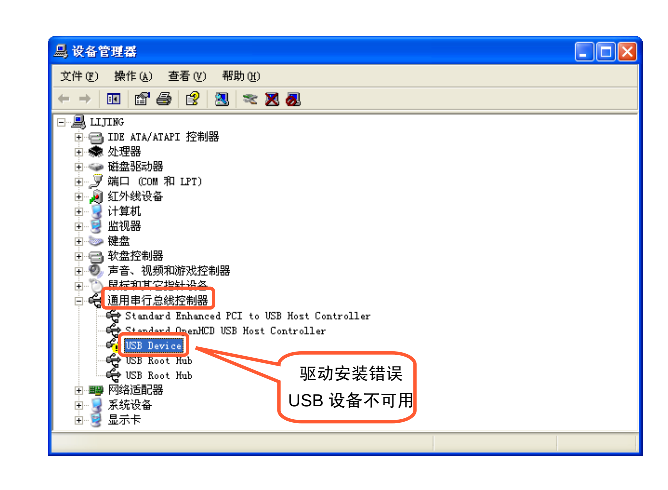


图 2.15 没有正确的安装新硬件结果

4

无法使用的 USB 设备需要重新安装驱动程序。点击鼠标右键，选择【**更新驱动程序**】选项，如图 2.16 所示。按照上述的过程重新安装驱动程序直到正确为止。

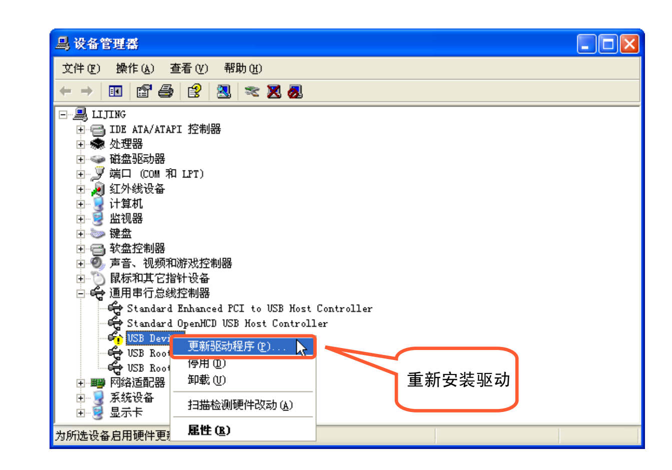


图 2.16 更新驱动程序

2.2 硬件连接方法

用户在使用 TKScope 仿真器进行仿真之前，需要把仿真器和 PC 机、用户目标板这些硬件设备正确的连接起来。



开始之前！

- TKScope 仿真器型号很多，用户在使用之前，根据实际仿真需要，选择合适的仿真器型号。
- TKScope 仿真头种类各异，用户在仿真之前，根据实际要仿真的芯片，选择合适的仿真头。

连接仿真器和计算机

TKScope 仿真器是用 USB 口通信的，用 USB 连接线把仿真器的 USB 口和计算机的 USB 口连接起来。

连接仿真器和目标板

TKScope 仿真器的仿真头采用"ECP＋POD 主板＋ADP"结构，用户根据实际要仿真的芯片选择合适的仿真头。然后，把仿真头插到目标板上预留的仿真接口。



举例：选择仿真头的方法。

例如，仿真 P89V51RD2FA 芯片，用户目标板预留 DIP40 的仿真接口插座。

此时，ECP 板需要选择 ECP-8051HS-PLCC44，上面的底座安装 P89V51RD2FA 芯片。

POD 主板需要选择 POD-8051HS-P84。

ADP 板需要选择 ADP-8051HS-DIP40。

仿真器上电

用出厂所配的电源适配器给仿真器上电，电源开关拨到 ON 位置，电源指示灯 PWR 呈红色点亮状态，CLK、BSY、ERR 交替闪烁数次后，BSY、ERR 熄灭，CLK 继续闪烁，之后 USB 指示灯呈蓝色点亮状态，此时仿真器就可以正常的工作了。



注意！必须使用出厂所配的电源适配器，其他规格的电源适配器可能会导致仿真器损坏。

2.3 在 Keil IDE 环境下的入门操作

本节主要讲解 TKScope 仿真器在 Keil IDE 环境下的使用流程和方法，请您拿起手中的仿真器，follow me 进行第一次入门操作吧！



本节主要讲解 Keil IDE 环境下与建立工程、仿真调试相关的各项功能的使用方法，用户若要学习使用关于 Keil 软件的更多功能，请查看 Keil 软件的帮助信息或查阅相关的书籍。

2.3.1 新建工程

本小节主要讲解在 Keil IDE 环境下建立新工程的步骤和方法，您如果有建立好的工程或对 Keil 软件的使用很熟悉，可以略过此节，直接跳到下一小节仿真环境的设置。



本小节主要讲解在 Keil IDE 环境下新建工程的方法，没有涉及到具体程序代码的编写方法和技巧，用户需查阅相关资料或书籍，编写自己的程序代码文件。

新建工程框架

1 双击 Keil 图标启动软件，即可进入 Keil 的主界面。选择【**Project**】菜单下的【**New Project**】选项，如图 2.17 所示，建立新的工程。

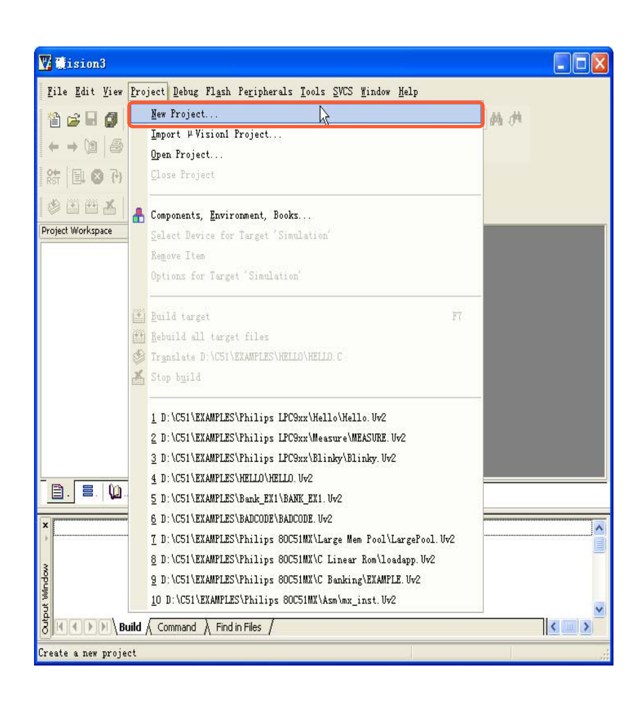


图 2.17 建立新工程

2 在弹出的新建工程对话框内，输入工程的名称，然后点击【**保存**】，如图 2.18 所示。工程的名称尽量与工程的性质、内容相关，增加可读性。

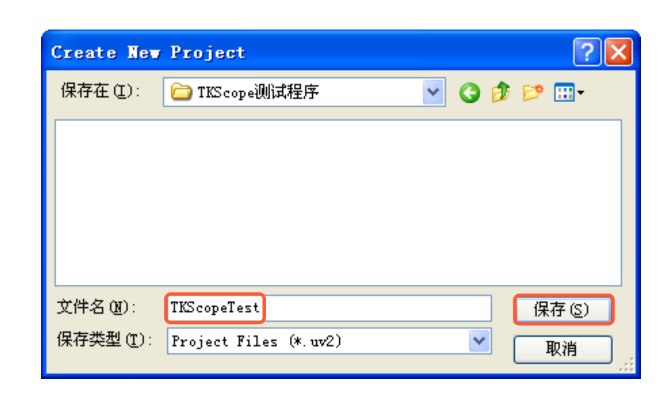


图 2.18 输入新建工程名称



Keil 开发环境是以工程的模式来管理文件的，用户应当把每个工程保存到独自的文件夹内。在计算机上新建一个文件夹，用于保存即将建立的工程。

3

系统弹出器件选择对话框，用户根据实际仿真的芯片型号选择器件，然后点击【**确定**】。本例中仿真的芯片是P89V51RD2FA，选择如图 2.19 所示。

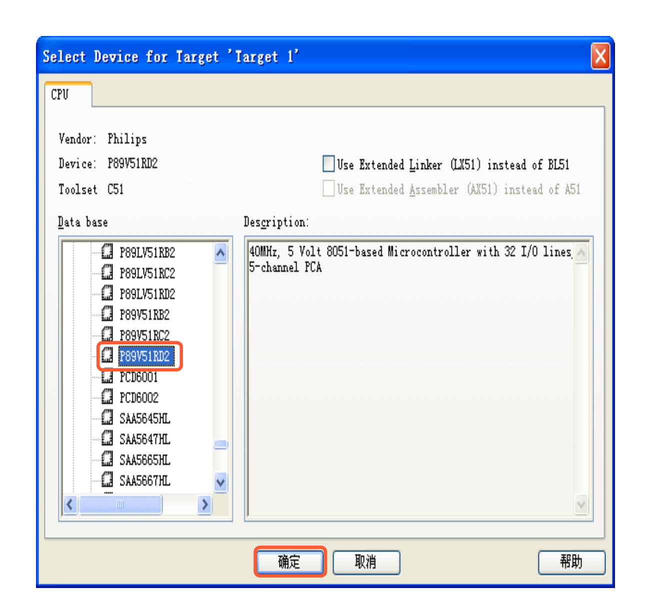


图 2.19　选择器件型号



如果开发环境里面没有实际仿真的芯片型号，采用相近原则，选择性能相近的型号就可以。

4

系统弹出对话框，如图 2.20 所示，用户可以选择【**是**】，添加 Startup 文件到工程中，也可以选择【**否**】，不添加此文件。这个文件对仿真器的仿真没有影响，一般情况下选择【**否**】即可。



图 2.20　添加 Startup 文件询问对话框

5

至此，新的工程框架已经建立起来了，但是里面还没有内容，接下来就是添加用户程序到工程中。

添加程序文件

1

选择【**File**】菜单下的【**New**】选项，如图 2.21 所示，新建程序文件。

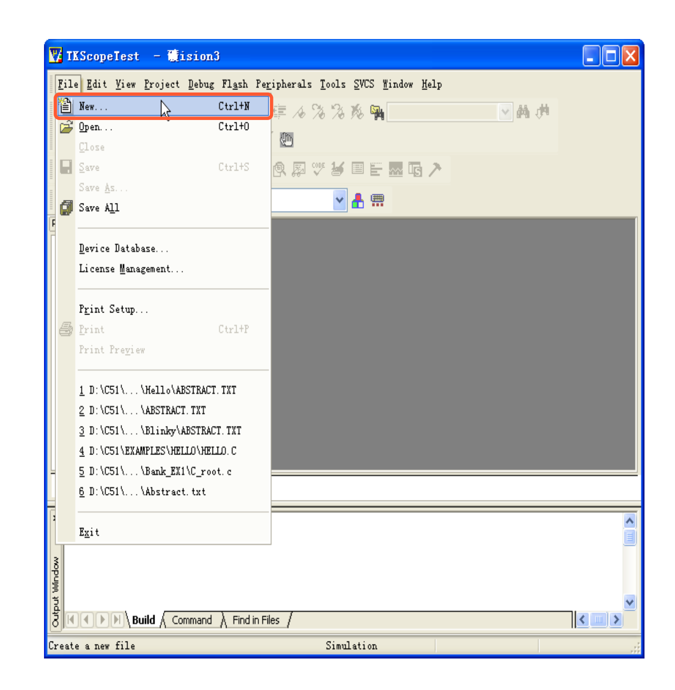


图 2.21　新建文件

2 系统弹出一个文本文件对话框。在此输入用户程序，如图 2.22 所示，然后点击【**保存**】。

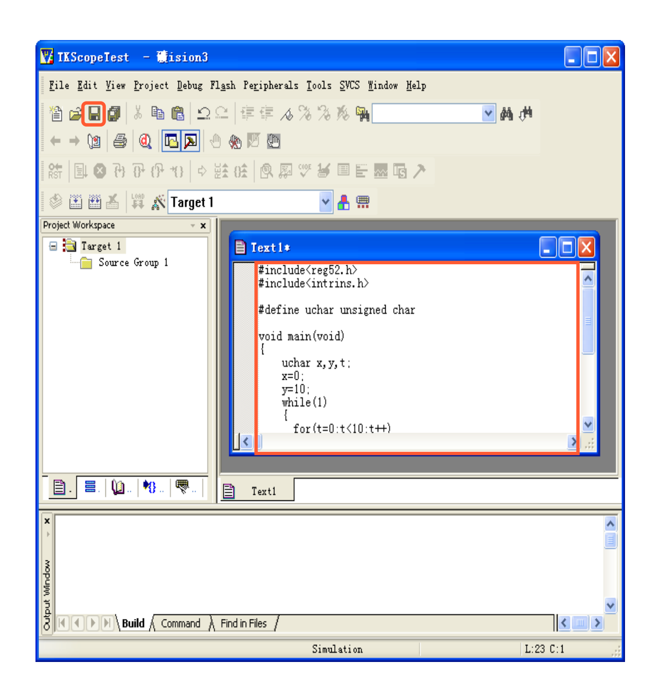


图 2.22　编写用户程序

3 系统弹出文件保存对话框。用户输入文件的名称和扩展名，如图 2.23 所示，然后点击【**保存**】。

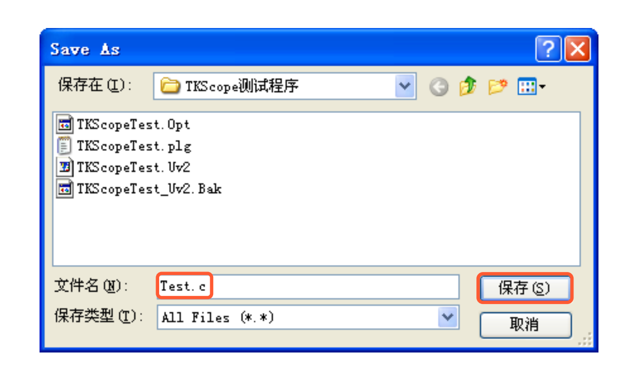


图 2.23　保存用户程序



文件的扩展名与程序的语言种类有关，程序采用汇编语言编写扩展名为.asm，程序采用 C 语言编写扩展名为.c。文件的名称尽量与程序的性质、内容相关，增加可读性。例程采用 C 语言编写，只是一个简单的测试程序，故保存为 Test.c。文件和工程要保存到同一个文件夹内。

4 在主界面左侧的工程窗口中，选中【**Target 1**】下面的【**Source Group 1**】选项，点击鼠标右键，在弹出的菜单中选择【**Add Files to Group**】选项，如图 2.24 所示，把用户程序文件添加到工程中。

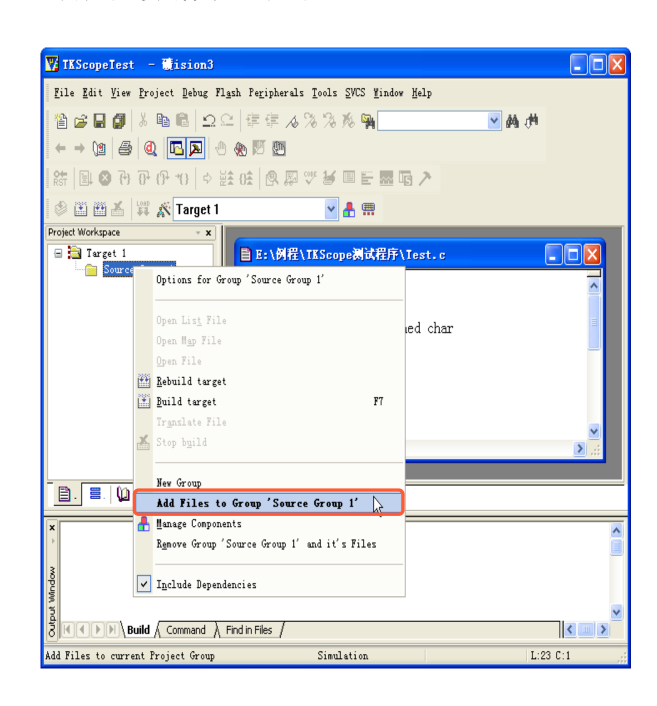


图 2.24　添加用户程序文件

5 在系统弹出的文件选择对话框内，选中要添加的程序文件如图 2.25 所示。然后，点击【**Add**】即可完成程序文件添加到工程中的操作。

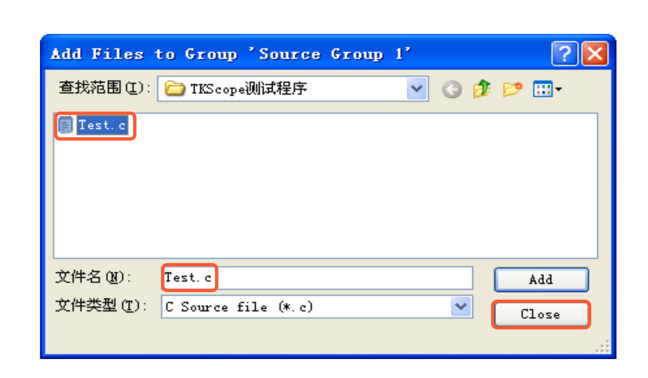


图 2.25　选择用户程序文件

6 至此，一个完整的工程已经建立起来。但还不能保证用户程序的正确性，需要通过系统链接、编译进行检测。

工程文件链接编译

1 选择【Project】菜单下的【Rebuild all target files】选项（如图 2.26 所示）或点击快捷图标 进行工程编译。结果在下面的信息输出窗口中可以看到。

2 用户根据编译结果的提示信息来纠正程序中的错误，直到编译完全通过没有任何错误为止。例程编译完全通过，程序没有任何错误，如图 2.27 所示，此时就可以使用仿真器进行仿真调试了。

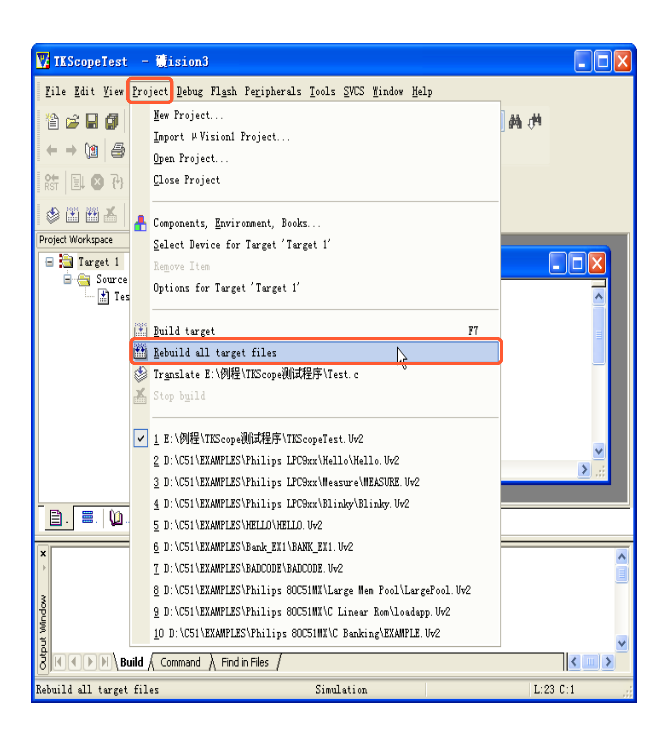


图 2.26　编译工程文件

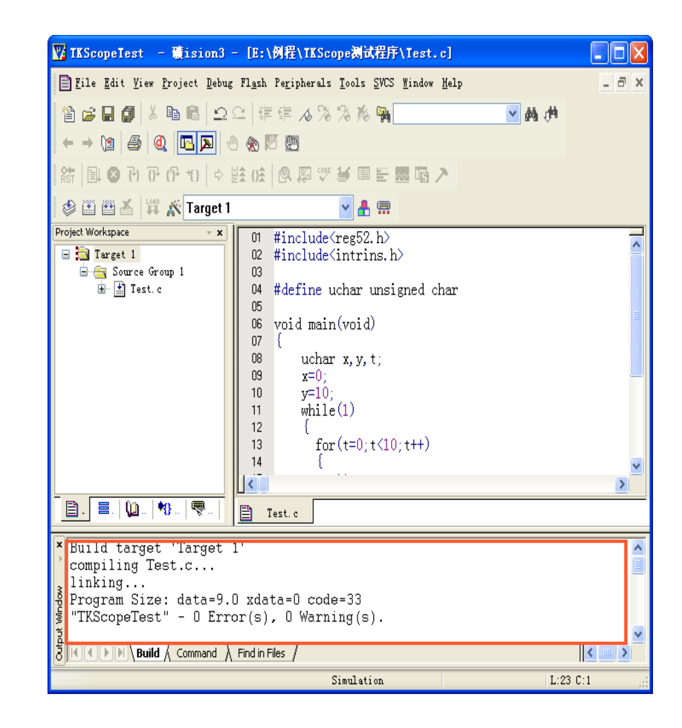


图 2.27　编译结果



注意！用户程序必须链接编译正确之后，才能使用仿真器进行硬件仿真。

2.3.2 仿真环境设置

工程链接编译正确之后，就可以使用 TKScope 仿真器进行实际的硬件仿真。



注意！在仿真之前还需要配置硬件仿真参数，否则可能引起仿真失败。

硬件仿真环境设置

1 选择【**Project**】菜单下【**Options for Target**】选项，或点击快捷图标 ，如图 2.28 所示，即可进入工程信息设置界面。

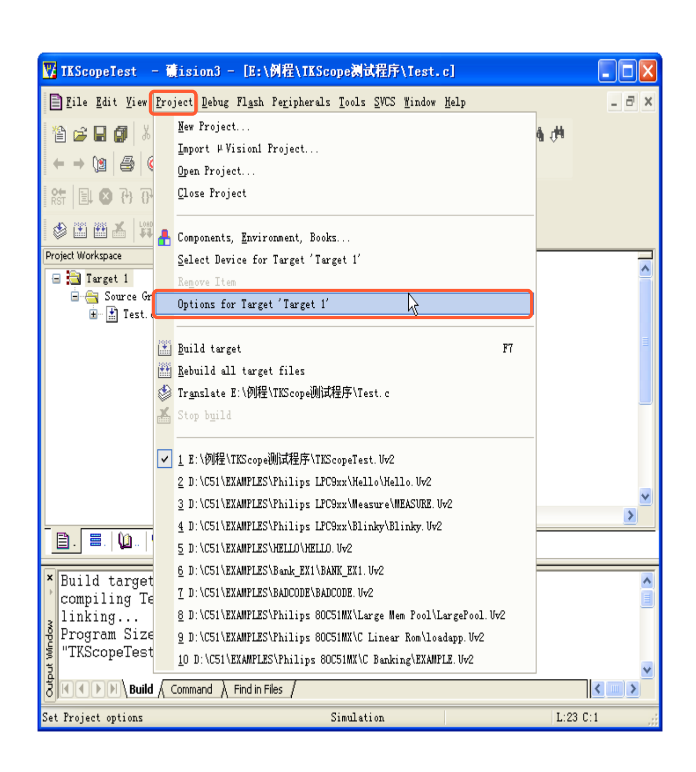


图 2.28 工程信息设置选项

2 工程信息设置界面如图 2.29 所示。

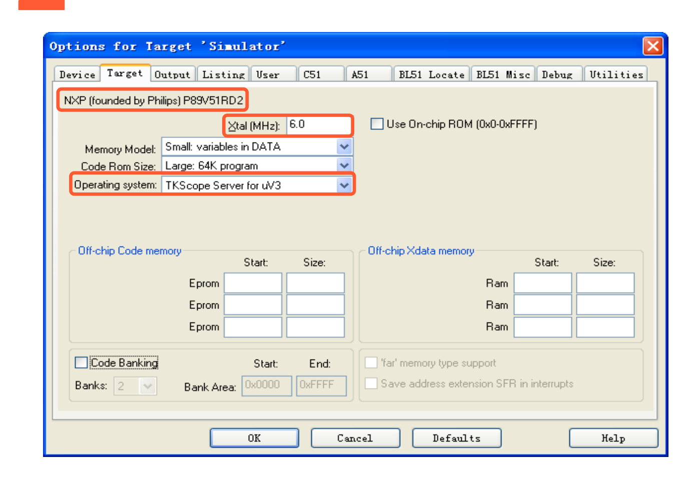


图 2.29 工程信息设置界面

【**Device**】选项是用于芯片选择。

在建立工程时没有选择芯片型号或选择不正确，在此可以进行选择，选择的芯片型号系统会显示出来，如图 2.29 所示。

【**Target**】选项主要注意两处选择，在图 2.29 中有所标注。一是【**Xtal**】时钟选项，这一项要和后面提到的仿真器硬件时钟设置一致，否则影响仿真精度，在后面仿真器的设置中会再次提到。二是【**Operating**】选项，这一项要和当前的 Keil 版本一致，如当前使用的是 Keil uV3，选项如图 2.29 所示。

【**Debug**】选项里面要选择右侧的硬件仿真，并选择正确的仿真器驱动，如图 2.30 所示。然后，点击【**Settings**】进入 TKScope 仿真器的设置界面，如图 2.31 所示。

工程信息设置界面里面的其它选项，如非特殊需要可以保持默认值不必修改。

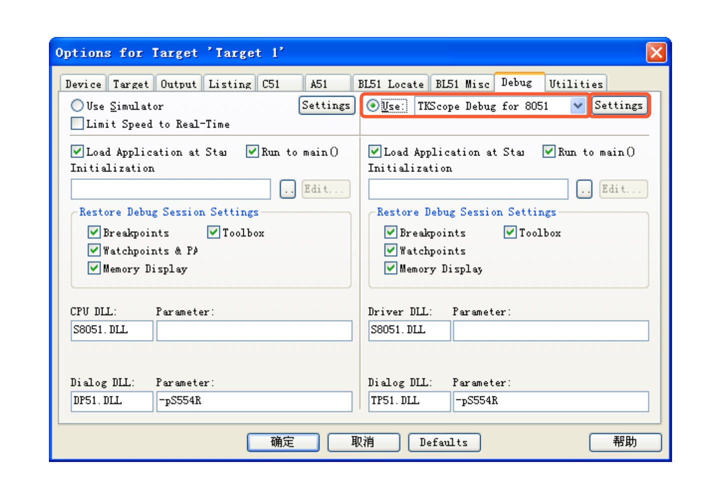


图 2.30 调试选项设置

仿真器工作参数设置

1 TKScope 仿真器的设置界面如图 2.31 所示。用户在第一次运行仿真器时，系统将采用缺省标准配置。缺省配置可以满足您的一般需求，如果不能满足，您需要手动进行设置，设置后的配置信息系统将予以保存，下次启动时可持续采用。

注意！ TKScope 仿真器在运行前需要进行参数配置，否则可能出现错误的运行结果。

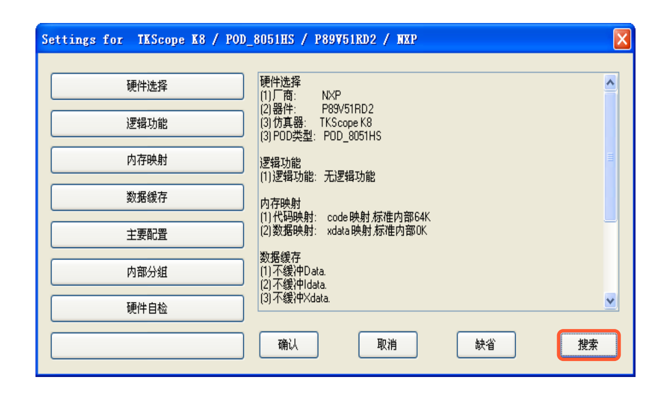


图 2.31 TKScope 仿真器设置界面



下面，简单介绍一下各项设置功能的界面和特别需要注意的地方，具体的使用方法和各个功能模块的含义用户请详见《第 3 章》。

2 点击【硬件选择】进入如图 2.32 所示的界面，此时必须正确选择当前仿真的芯片型号和仿真器类型。用户可以根据实际情况手动进行硬件选择，如图 2.32 所示；也可以依靠系统的搜索功能。

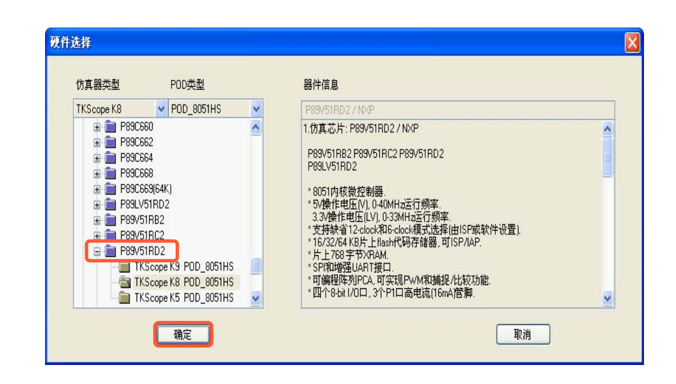


图 2.32 硬件选择选项



使用搜索功能时，用户只需正确选择仿真芯片的型号，不必指定仿真器具体类型，然后点击【**确定**】返回到图 2.31 的界面，点击【**搜索**】选项，系统会自动搜索到当前仿真器的类型，用户保存配置信息即可。



注意！ 对于 TKScope 仿真器里面没有实际仿真的芯片型号，采用替代原则，选择功能相近的型号替代选择，但是仿真头上面安装的芯片必须是实际要仿真的芯片。

3 点击【逻辑功能】进入如图 2.33 所示的界面，缺省状态是无逻辑功能。各项逻辑功能根据用户实际仿真需要来选用，具体功能讲解请详见《第 3 章—逻辑功能》。

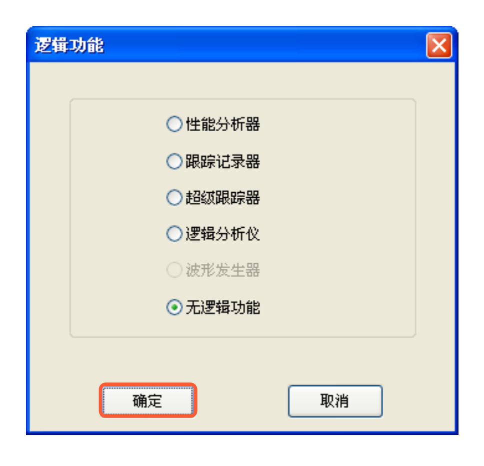


图 2.33 逻辑功能选项

4

点击【内存映射】进入如图 2.34 所示的界面，图中状态是缺省状态，也是比较常用的状态。用户可以根据实际仿真需要添加 xdata 空间和 code 空间映射情况，具体功能讲解请详见《第 3 章—内存映射》。



图 2.34　内存映射选项

5

点击【数据缓存】进入如图 2.35 所示的界面，图中状态是缺省状态。缺省状态下只选中 code 缓存，实际上也建议用户只选中 code 缓存。具体功能讲解请详见《第 3 章—数据缓存》。

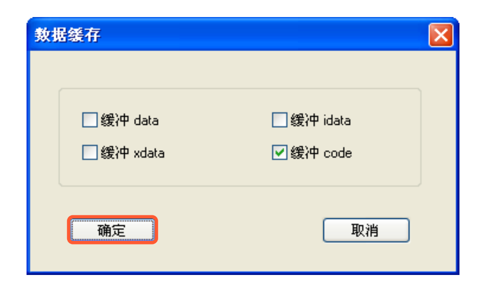


图 2.35　数据缓存选项

6

点击【内部分组】进入如图 2.36 所示的界面，图中状态是缺省状态，也是比较常用的状态。具体功能讲解请详见《第 3 章—内部分组》。

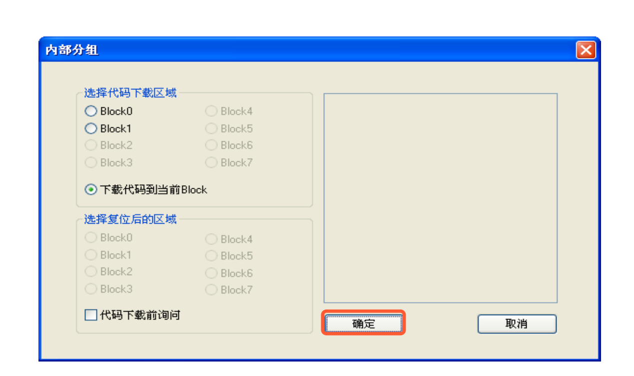


图 2.36　内部分组选项

7

点击【主要配置】进入如图 2.37 所示的界面，在此时钟设置这项要特别注意。前面也提到过这个问题，图 2.29 中也有一处时钟设置，这两个时钟值要设置一致，才能保证仿真的精度。

例如，图 2.29 中的时钟值设置为 6MHz，这里选用内部时钟选项，时钟值也要设置为 6MHz。其它各项的设置没有特殊要求，按照实际需要进行配置就可以，具体功能讲解请详见《第 3 章—主要配置》。



图 2.37　主要配置选项

8

点击【硬件自检】进入如图 2.38 所示的界面。此时点击【开始】选项可以对仿真器及其连接的组件进行硬件自检，在窗口中可以看到自检的结果信息。硬件自检是 TKScope 仿真器非常重要的功能，用于仿真器出现问题时进行自身检测找到故障原因，解决了用户在使用仿真器过程中，出现莫明其妙的问题时束手无策的尴尬局面。

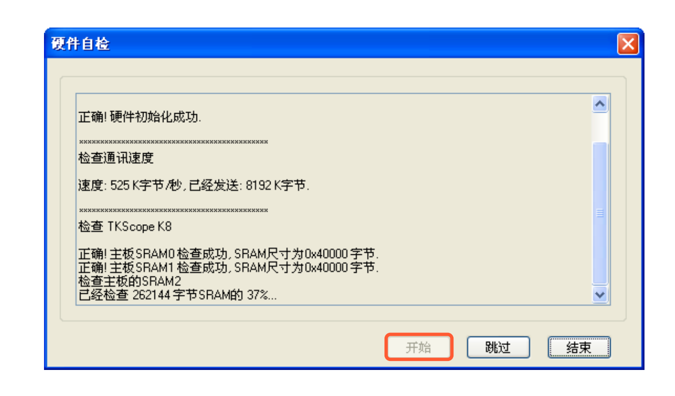


图 2.38　硬件自检过程

2.3.3 仿真调试

硬件仿真环境和仿真器工作参数根据实际需要设置完成之后，点击确认，接下来就可以使用 TKScope 仿真器进行硬件仿真调试了。

进入仿真调试状态

进入调试仿真状态的方法很简单，选择【**Debug**】菜单下的【**Start/Stop Debug Session**】选项，如图 2.39 所示，或点击快捷图标 即可进入调试仿真状态。

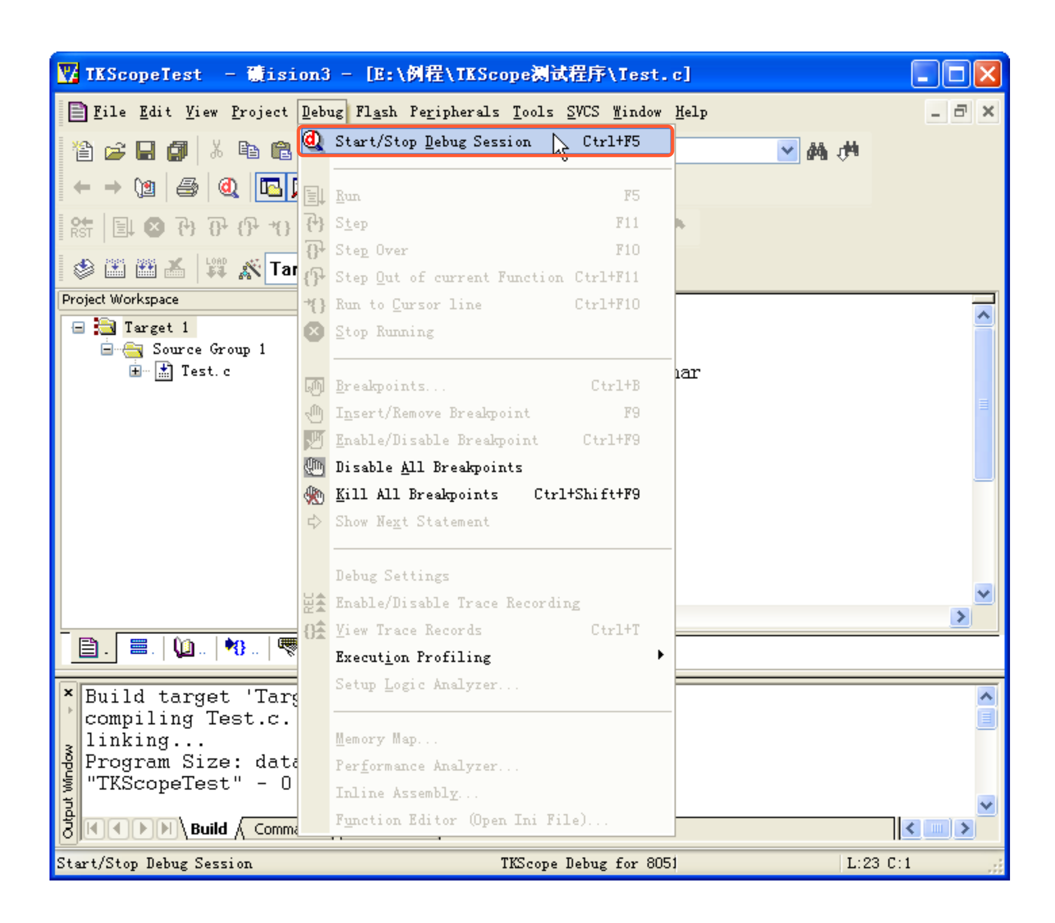


图 2.39　进入调试仿真状态

进入调试状态后，用户可根据自己的实际仿真情况需要，全速运行、设置断点或单步执行等，在此过程中，可以打开相应的窗口查看寄存器和变量的值，便于用户分析程序。

全速运行程序

选择【**Debug**】菜单下的【**Run**】选项或点击快捷图标 或按动键盘 F5，可以全速运行用户程序。

TKScope 仿真器具有 On the fly 运行中操作功能，允许用户在程序运行中查看修改全部资源。打破了以往仿真器程序运行中不能查看任何资源不能接收任何命令的局面，极大的方便了用户分析程序的运行。

On the fly 运行中操作功能的使用方法，用户请详见《**第 7 章—On the fly 运行中操作**》。

断点操作

用户可以在程序中设置断点，让程序运行到指定位置停止下来，查看分析程序的当前状态。

TKScope 仿真器共有 5 种断点，程序运行断点、时间断点、MOVC 代码读取断点、MOVX 数据读取断点、MOVX 数据写入断点。关于断点的具体操作方法，用户请详见《**第 5 章—断点操作技巧**》。

单步调试

选择【**Debug**】菜单下的【**Step**】选项或点击快捷图标 或按动键盘 F11，可以单步跟踪运行用户程序，尽最大的可能跟踪当前程序的最小运行单位。在 C 语言环境下调试可以跟踪到每一个 c 程序行，在汇编语言环境下调试可以跟踪到每一个汇编指令的执行。

选择【**Debug**】菜单下的【**Step Over**】选项或点击快捷图标 或按动键盘 F10，可以单步运行用户程序，尽最大的可能执行完当前的程序行。与 F11 最大的不同是，在 C 语言环境下调试不跟踪进入子函数内部。

运行到光标处

用户也可以使用运行到光标处功能，即程序运行到光标所在程序位置时停止运行，调试程序非常方便。鼠标点击用户希望停止运行的程序行，把光标移动此程序行，然后选择【**Debug**】菜单下的【**Run to Cursor line**】选项或点击快捷图标 ，程序运行到光标处即可停止下来。

退出仿真调试状态

仿真结束后，即可退出调试状态，方法和进入调试状态的方法一样，选择【**Debug**】菜单下的【**Start/Stop Debug Session**】选项，如图 2.32 所示，或点击快捷图标 。至此，就是一个完整的使用 TKScope 仿真器仿真调试程序的过程。



第 3 章

TKScope 仿真器功能模块介绍

3.1 功能设置主界面



开始之前！

- TKScope 高级专业仿真器，是一款功能强大的通用仿真器，可仿真芯片数目种类达数千种。
- 用户在使用之前必须正确合理的设置，仿真器才能淋漓尽致的发挥它的功能，达到您所要的理想的仿真效果。
- 本章主要讲解 TKScope 仿真器各项功能模块的含义和使用方法，让用户更好的了解 TKScope 仿真器的各项功能，以便根据实际仿真需要更加合理的使用 TKScope 仿真器。

用户选择 TKScope 硬件仿真，点击设置选项后即可进入 TKScope 仿真器的设置主界面，如图 3.1 所示。

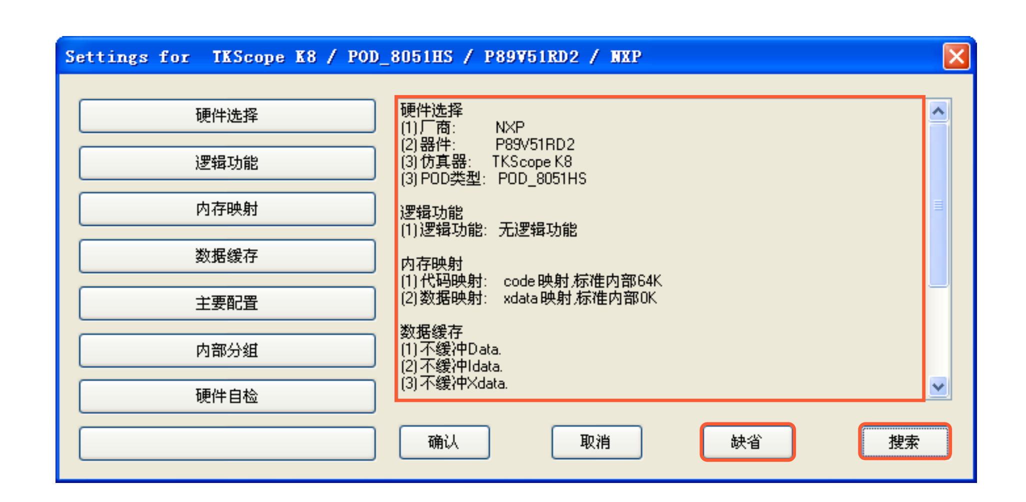


图 3.1　TKScope 仿真器设置主界面

TKScope 仿真器内部的配置选项，在图 3.1 右侧的信息提示框中可以看到。TKScope 仿真器具有记忆功能，进入后是用户上次使用时的配置值，初次使用时是出厂的缺省配置值。用户可以点击【**缺省**】选项恢复出厂时的配置值。

主界面中的【**搜索**】选项，实用性很强，用来系统自动搜索仿真器型号和 POD 头型号，主要配合硬件选择使用，在下一小节《**硬件选择**》中具体讲解使用方法和注意事项。

3.2 硬件选择

在 TKScope 仿真器设置主界面中，点击【**硬件选择**】选项，进入如图 3.2 所示的界面。

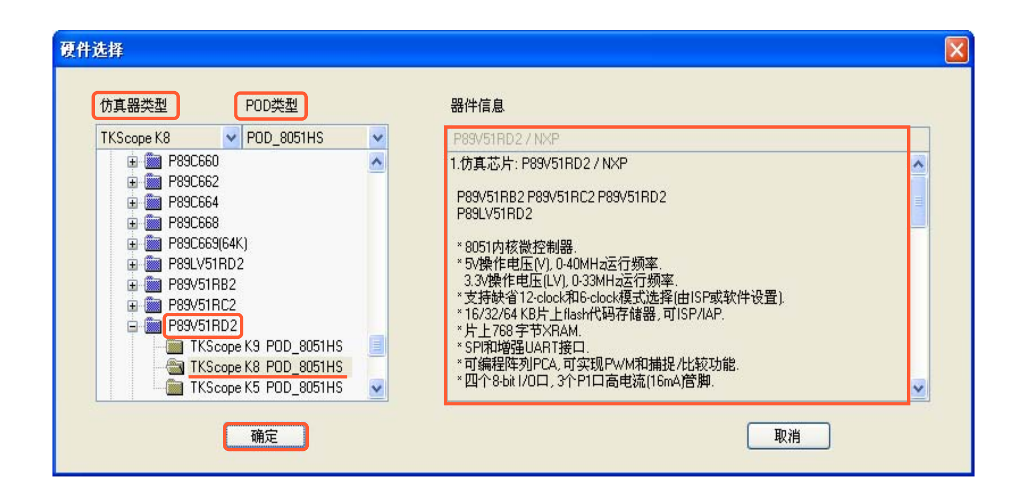


图 3.2　硬件选择选项



TKScope 仿真器功能强大，能够仿真众多型号的 MCU，而且型号不断增加。因此，用户在使用前，必须正确选择芯片型号和仿真器硬件类型。

TKScope 仿真器的硬件采用【**仿真器＋POD 适配器＋芯片**】的结构，用户可以根据自己的仿真器硬件情况和实际仿真的芯片型号加以选择。

仿真器类型的选择

在图 3.2 左侧【**仿真器类型**】的下拉菜单中，选择当前的仿真器型号，具体型号在仿真器外壳上有明显的标注。

POD 适配器类型的选择

在图 3.2 左侧【**POD 类型**】的下拉菜单中，选择当前连接仿真器主机的 POD 适配器型号，具体型号在 POD 适配器的实体上有明显的标注。

仿真芯片的选择

仿真芯片型号的选择，通过图 3.2 中左侧的树状结构来完成。

芯片型号按照厂家进行排序，用户可以先找到芯片的厂家，然后点开该厂家的目录夹，找到符合要求的芯片型号。用户选择完成之后，在图 3.2 中右侧的【**器件信息**】中可以看到选择的提示信息。



注意！ 对于 TKScope 仿真器里面没有实际仿真的芯片型号，采用替代原则，选择功能相近的型号替代选择，但是仿真头上面安装的芯片必须是实际要仿真的芯片。

系统自动搜索功能

用户可以根据实际情况手动进行硬件选择，也可以依靠系统进行自动搜索。

自动搜索的方法很简单，用户只需正确选择仿真芯片的型号，如仿真 P89V51RD2 芯片，选择如图 3.2 所示，点击【**确定**】选项返回到图 3.1 的界面，然后点击【**搜索**】选项，系统会自动搜索到当前的仿真器和 POD 头型号以及内带的逻辑分析仪种类，弹出图 3.3 所示的搜索结果，用户选择【**是**】即可保存仿真器的硬件配置结果。

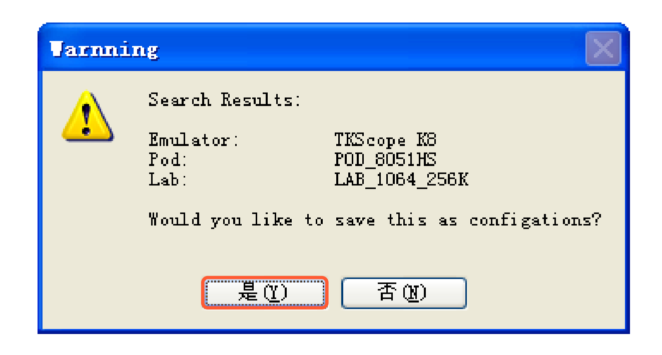


图 3.3　自动搜索仿真器配置结果



建议用户使用系统自动搜索功能，既快速又准确。

3.3 逻辑功能

在 TKScope 仿真器设置主界面中，点击【逻辑功能】选项，进入如图 3.4 所示的界面。

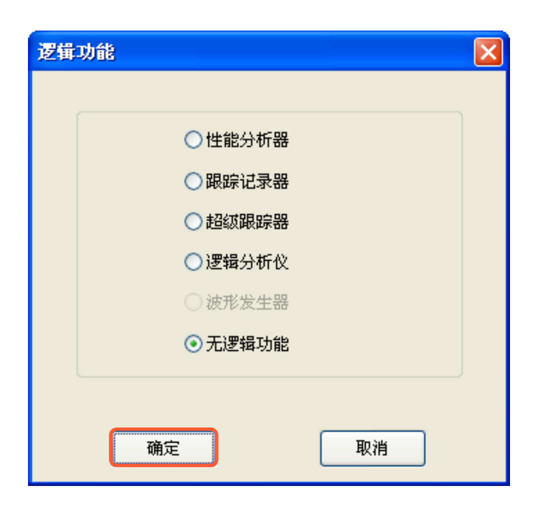


图 3.4　逻辑功能选项



TKScope 仿真器内置的 LAB 逻辑板具有繁多而强大的功能，用户可以根据自己在仿真时的实际情况加以选择。

性能分析器

性能分析（Performance Analyzer）是用来分析代码和数据的执行效率，并以统计图形的方式表现出来。

TKScope 仿真器内置的逻辑板可以同时完成代码和数据的性能分析。根据硬件的配置，最大可以同时进行 512KB 代码和 512KB 数据的性能分析。

性能分析器是 TKScope 仿真器的一项功能强大的仿真分析功能，结合具体的实例讲解才能更好的体现它的功能，在此只作上述的简单介绍，针对性能分析器有单独的篇幅详细的讲解使用方法，具体请详见《**第 6 章—性能分析器的使用方法**》。

跟踪记录器

跟踪记录（History Trace）是用来实时记录运行的程序轨迹，用于分析程序的运行流程。

TKScope 仿真器除记录程序运行轨迹外，还以 32 位容量记录每步的消耗时间。根据程序运行轨迹和时间消耗轨迹，用户可以更完整的分析程序运行。根据硬件的配置，最大可同时记录 512KB 步的代码运行和消耗时间轨迹。

超级跟踪器

超级跟踪（Super Trace）为独创的增强性跟踪记录，极大提高了跟踪记录的实用性。

在该模式下，除保留了 History Trace 的全部功能外，还可实现跟踪记录芯片内部的主要资源，如 ACC/B/DPTR/SP/R0-7 等。根据不同的芯片和硬件，可记录的资源情况可能不同。

跟踪记录功能是仿真中非常实用的功能，TKScope 仿真器更是独具特色，不仅记录程序的运行轨迹，还同步记录程序的消耗时间，超级跟踪更是强悍增加了芯片内部主要资源的记录。跟踪功能结合具体的实例讲解才能更好的体现它的强悍功能，在此只作上述的简单介绍，针对超级跟踪和跟踪记录功能有单独的篇幅详细的讲解使用方法，具体请详见《**第 6 章－超级跟踪功能的妙用**》。

逻辑分析仪

TKScope 仿真器内置的 LAB 逻辑板在仿真的同时，还可独立运行**最大 64 路的逻辑分析功能**。64 路分 32 路内部仿真信号和 32 路外部信号，逻辑分析的启动和停止，可以与仿真器的状态相关联，也可以独立的作为逻辑分析仪来使用。

逻辑分析仪是电子工程师开发过程中的好帮手，越来越广泛的应用到数字电路开发中，其集成逻辑分析、总线分析、协议分析、频率计、逻辑笔等众多功能于一身，让您无需火眼金晶就能轻松发现隐藏在系统中的错误。TKScope 仿真器将仿真和逻辑分析功能结合在一起，让您的程序仿真状态来控制逻辑分析的启动和停止，这样您可以更加方便的排查系统中存在的错误和隐患。

逻辑分析仪功能是 TKScope 仿真器非常重要的功能模块，可以独立的作为逻辑分析仪使用，使用方法和独立的成品逻辑分析仪相同，可以参考逻辑分析仪的用户使用手册，也可以和仿真器配合起来使用，使用方法我们将以单独的篇幅给出，具体请详见《**第 6 章－逻辑分析仪的使用方法**》。

无逻辑功能

无逻辑功能就是只当作仿真器使用，没有上述的增强型逻辑仿真功能，用户无需性能分析、跟踪记录、逻辑分析的情况下选择无逻辑功能即可。

3.4 内存映射

在 TKScope 仿真器设置主界面中，点击【内存映射】选项，进入如图 3.5 所示的界面。

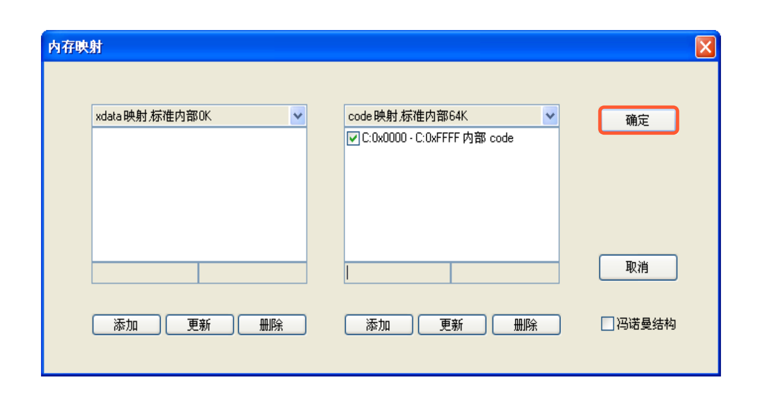


图 3.5　内存映射选项

TKScope 仿真器内部有最大 512KB 的代码空间/最大 512KB 的数据空间供用户使用，用户也可以使用全部或部分的外部程序空间/数据空间，影像的分辨率为 1Byte，用户可以把任一地址的程序/数据分配到仿真器内部或外部。

代码空间影像

TKScope 仿真器允许用户设置内部代码空间的影像，代码空间影像 Code Memory Map 窗口中已经为用户设置了几个标准配置，分别是内部 64K/32K/16K/8K/4K/0K，则外部代码空间为 0K/32K/48K/56K/60K/64K。如果这些标准设置不能满足您的要求，用户可以添加自己的影像范围。



注意！添加的影像是内部代码空间，没有涉及到的空间地址则影像到仿真器外部。

1

添加影像范围的方法很简单，用户只需输入影像范围的起始地址和结束地址即可。

例如，用户添加 2K（0x0000～0x07FF）影像范围，只需按照图 3.6 所示输入，然后点击【**添加**】即可。

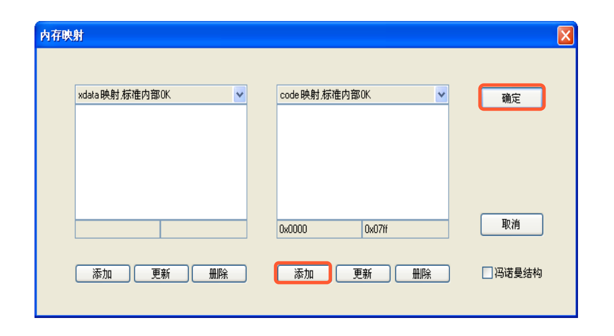


图 3.6　用户定义影像范围

2

用户如果要改变添加的影像范围可以重新输入起始地址和结束地址，然后点击【**更新**】即可。

例如，用户想要更改为 1K（0x0000～0x03FF）影像范围，只需按照图 3.7 所示，更改输入，然后点击【**更新**】即可。



图 3.7　用户更改自定义的影像范围

3

用户如果要取消添加的影像范围，选中之后点击【**删除**】即可取消，如图 3.8 所示。

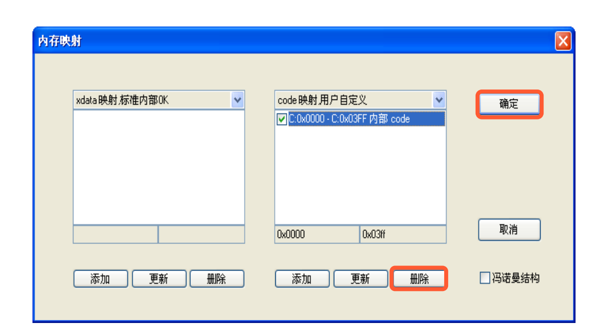


图 3.8　用户取消自定义的影像范围

数据空间影像

TKScope 仿真器允许用户设置内部数据空间的影像，数据空间影像 Xdata Memory Map 窗口中已经为用户设置了几个标准配置，分别是内部 64K/32K/16K/8K/4K/0K，则外部数据空间为 0K/32K/48K/56K/60K/64K。

如果这些标准设置不能满足您的要求，用户可以添加自己的影像范围。数据空间影像范围的自定义操作方法同代码空间影像范围的自定义操作方法相同，这里不再重复讲解。



注意！添加的影像是内部数据空间，没有涉及到的空间地址则影像到仿真器外部。



内部数据空间同外部数据空间的区别！

内部数据空间同外部数据空间在使用上是完全相同的。但是，如果在 Use No Bus 下使用内部数据空间，由于 P0/P2 口不输出地址/数据，因此内部的 64KB 数据空间类似于片上 xdata，此时 wr/rd 信号线将正常输出信号。



注意！

如果某一个地址的 xdata 影像到仿真器内部，则该地址外部的 xdata 被忽略。

如果用户外部有一些 xdata 的外设，不要将该外设地址映射到仿真器内部。

冯.诺曼结构

TKScope 仿真器内部的 64KB 数据空间支持冯.诺曼结构，该结构既可以当作程序运行又可以当作数据空间进行操作。在添加数据影像时，选中冯.诺曼结构添加的项目具有冯.诺曼结构。

3.5 数据缓存

在 TKScope 仿真器设置主界面中，点击【**数据缓存**】选项，进入如图 3.9 所示的界面。



图 3.9　数据缓存选项

数据缓存（Cache Options）配置主要是解决屏幕刷新和仿真速度的矛盾。

如果用户选择 Cache，屏幕显示刷新只在用户程序运行后进行，这样可以加快显示速度，但是如果某一个操作引起的其它数据的变化可能不能及时显示，刷新只能发生运行用户代码以后。

如果用户不选择 Cache，则用户在 PC 端软件的任何操作都将引起显示数据的重新刷新，在查找不稳定硬件时比较理想，但屏幕刷新会影响操作响应速度。

1	**Cache Data**	直接寻址 Data(SFR)缓存
2	**Cache Idata**	间接寻址 Idata 缓存
3	**Cache Xdata**	外部数据 Xdata 缓存
4	**Cache Code**	代码数据 Code 缓存


建议用户使用缺省 Cache 设置，仅选择 Cache Code。

3.6 主要配置

在 TKScope 仿真器设置主界面中，点击【主要配置】选项，进入如图 3.10 所示的界面。



图 3.10　主要配置选项



注意！ TKScope 仿真器在运行前需要进行参数配置，否则可能出现错误的运行结果。



您在第一次运行仿真器时，系统将采用缺省标准配置。缺省配置可以满足您的一般需求，如果不能满足，您需要手动进行设置，设置后的配置信息系统将予以保存，下次启动时可持续采用。

辅助设置

辅助设置为您提供一些特殊的仿真要求，在一般情况下这些选项的缺省设置能保证用户程序的运行。

1 初始代码

仿真器在调入用户程序前将初始化内部程序数值。代码下载后，没有用户代码的位置仍保留初始化数值。这种功能在一些特殊场合非常有用。

2 外部复位

仿真时如果使用外部复位输入，用户板复位脚的信号将影响用户程序的运行。程序运行时，如果发现外部复位信号，程序将复位后继续运行。

3 外部电源

一般情况下，仿真器内部的仿真芯片使用的是内部稳定的 5V/3.3V/1.8V 电源，能确保最佳仿真性能。如果用户系统使用的电源和 5V/3.3V/1.8V 电源有差别，可以选择使用外部电源。外部电源从用户板的电源管脚输入，电压范围在 1.8V-5.5V 之间，用户必须保证该电源的稳定性。另外，由于 TKScope 仿真器内部可以更换仿真芯片，因此用户还必须了解外部输入电压是否在仿真芯片的正常工作范围内。

总线设置

为了取得最佳的仿真效果，总线分成三种仿真模式，用户可以根据实际情况加以选择，总线模式的选择需要同内存映射（Internal Memory Map）配合使用。

1 不用外部总线 No Bus

在仿真时，对于操作总线的情况都加以制止，主要针对 P0/P2 口。P0/P2 口在这种情况下是标准的 I/O 模式，遇到 MOVX/MOVC 指令时，P0/P2 口仍然表现出 I/O 特性而不输出总线信号。
如果用户没有使用总线则选择这种模式较好。另外，如果用户仅使用了 MOVX 指令来操作一些单片机的内部 xdata 空间，则也可以选择这种模式。

2 仅用数据总线 Only Xdata Bus

在仿真时，P0/P2 口将根据指令的运行情况来决定工作模式。遇到 MOVX 指令时，P0/P2 口将输出总线地址/数据信号，MOVX 指令结束后将恢复 I/O 模式。
如果用户使用 MOVX @Ri 指令，P2 口仍将表现出 I/O 特性，这种情况下 P2 口仍然可以作为 I/O 口使用。

3 自动判断总线 Auto Bus

在自动总线模式下，仿真器将根据用户的程序运行情况自动决定是否打开总线。
该模式在仿真内部/外部 xdata 地址重叠情况下非常有用，用户操作内部 xdata 时总线并不打开，但是在访问外部 xdata 时总线将自动打开。

4 使用全部总线 Use All Bus

在仿真时，P0/P2 口完全作为总线使用，用户可以通过 MOVX 指令操作仿真器外部数据空间，也可以通过 MOVC 指令读取外部代码数据，甚至运行外部的程序。
在这种模式下，P0/P2 口不能再作为 I/O 口使用。

时钟设置

TKScope 仿真器允许使用内部提供的时钟，也可以使用外部提供的时钟或晶振。仿真器内部提供的时钟由于进行了专门处理，在稳定性上要超过外部时钟，而且可以运行到更高的仿真频率。外部时钟主要是由用户板上的时钟提供，或通过仿真头上的振荡组件结合仿真头上的晶振或用户板上的晶振产生，由于存在不确定因素，可能会导致外部时钟不稳定，建议用户尽可能使用仿真器内部提供的时钟。

1 使用内部时钟

TKScope 仿真器内部提供超高精度的 PLL 连续可调时钟，时钟频率范围为 **20KHz-50MHz**，用户可以在时钟频率文本框中直接输入时钟频率值(MHz 单位)。

2 使用外部时钟

TKScope 仿真器可以使用仿真器外部提供的时钟，时钟范围为 **2MHz-24MHz**，这里的外部时钟是指通过仿真头上的振荡组件结合仿真头上的晶振或用户板上的晶振产生的时钟，用户在时钟频率文本框中直接输入外部时钟的值（MHz 单位）。

3 使用用户时钟

TKScope 仿真器也可以使用用户时钟，时钟范围为 **2MHz-24MHz**。
用户时钟是指用户目标板提供的时钟直接接到 X1 引脚上。



注意！用户时钟用指户目标板上的独立时钟，而不是用户目标板上的晶振产生的时钟。

断点设置

这里的断点是指当仿真器外部输入信号满足设置条件时，仿真器停止运行，可以说是硬件上的断点。

1 逻辑分析触发后停止

用户使用 TKScope 仿真器的逻辑分析仪功能时，选中此选项，则逻辑分析仪触发后用户程序停止运行。

2 S0、S1 满足条件后停止

S0、S1 为 TKScope 仿真器的外部输入信号，在机身上有 S0、S1 的输入端。用户可根据实际仿真需要接入外部信号到 S0、S1 端。用户可以设置 S0、S1 单独满足条件后停止，也可以设置组合方式满足条件后停止。

例如，用户希望 S0、S1 同时为低电平时，程序停止运行，则 S0 选择低电平，S0、S1 逻辑关系选择与，S1 选择低电平。

3.7 内部分组

在 TKScope 仿真器设置主界面中，点击【**内部分组**】选项，进入如图 3.11 所示的界面。



图 3.11　内部分组选项

TKScope 仿真器的 K9 型号独有 Bank 仿真功能！

- 突破 8051 的 64KB 代码数据限制，可仿真最大 8 分组的 64KB 空间。
- 支持代码分组（Bank）调试，最大 8×64KB。
- 支持数据分组（Bank）调试，最大 8×64KB。
- 支持 4 路仿真器外部 Bank 控制信号输入。
- 支持无限制数量 MCU 内部 Bank 控制信号。



内部分组选项用于 MCU 内部 Bank 分组信号的控制。关于 Bank 功能的原理、应用及外部扩展 Bank 模块的仿真方法，具体请详见《**第 8 章—Bank 功能的使用方法**》。

1 选择代码下载区域

用户可以根据实际仿真需要选择程序下载到相应的 Block 区。

如 P89V51RD2 芯片内部有 Block0、Block1 两个区，用户可以指定程序下载到某个 Block 区域。

2 下载代码到当前 Block

有些种类的芯片，上电复位后自动指定程序下载的 Block 区域。用户也可以选择这种方式，将程序下载到当前的 Block 区域内仿真。

如 P89V51RD2 芯片复位后指定的区域是 Block1，用户如果选择这种下载方式，程序就是下载到当前的 Block1 区内进行仿真。

3 代码下载前询问

选中此项，进入监控状态前装载用户程序时，系统会弹出如图 3.12 所示的对话框，用户可以看到当前设置的 Block 区域选择情况。此时，用户可以进行修改，重新选择代码下载的 Block 区域。

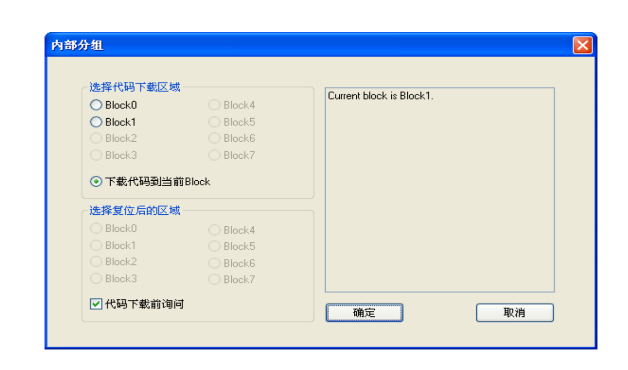


图 3.12　代码下载询问框

3.8 硬件自检

在 TKScope 仿真器设置主界面中，点击【硬件自检】选项，进入如图 3.13 所示的界面。

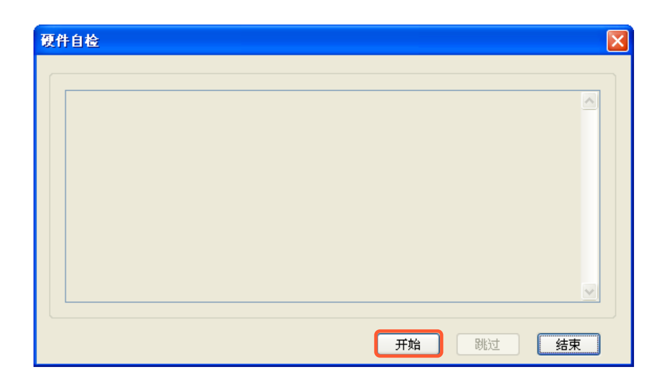


图 3.13 硬件自检选项

硬件自检主要用于 TKScope 仿真器自身功能检测以及连接组件故障检测。点击图 3.13 中的【开始】选项，即可开始硬件自检，此时自检过程状态如图 3.14 所示。

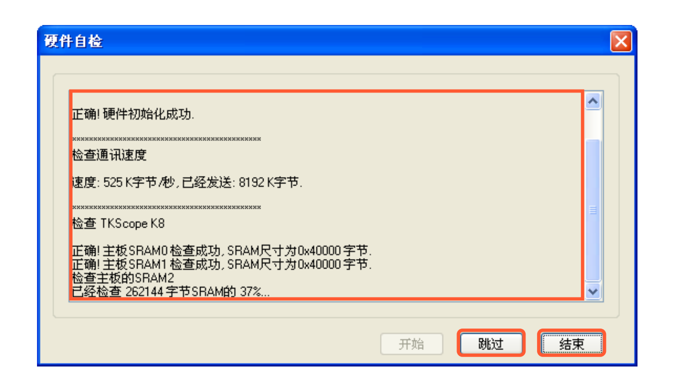


图 3.14 硬件自检状态

硬件自检过程中，用户可以点击【跳过】选项，跳过某项功能的自检过程；也可以点击【结束】选项，提前结束自检过程。硬件自检 100％全部正确通过的结果，如图 3.15 所示。

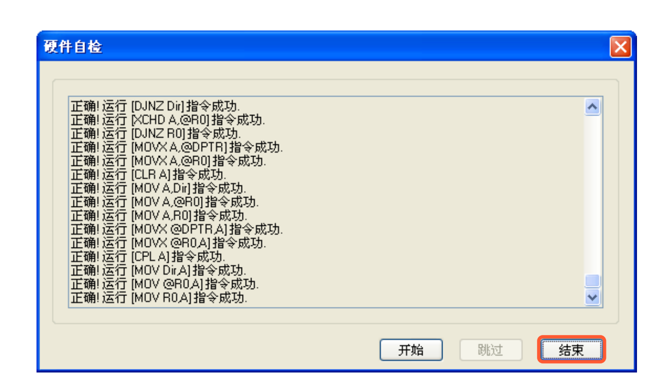


图 3.15 硬件自检全部正确完成

TKScope 仿真器硬件自检流程：

- 检查硬件初始化
- 检查 USB 通讯速度
- 检查当前 TKScope 仿真器型号
- 检查主板 SRAM
- 检查逻辑分析仪
- 检查硬件复位
- 检查代码映射
- 检查数据映射
- 检查 XRAM

硬件自检过程中，用户在信息提示框中（如图 3.14 所示）可以清楚的看到仿真器自检结果的提示信息。

如 USB 通讯速度，用户在提示框中可以看到每秒钟发送的字节数和发送的全部字节数，这样就能够直观的感受到 TKScope 仿真器的高速通讯速度。

在硬件自检的过程中，用户可以看到仿真器各个功能器件的自检结果，如硬件初始化、复位等功能是否正常，SRAM、XRAM 等器件是否正常工作以及内部是否有损坏，代码映射、数据映射空间是否正常工作以及内部是否有损坏等等。

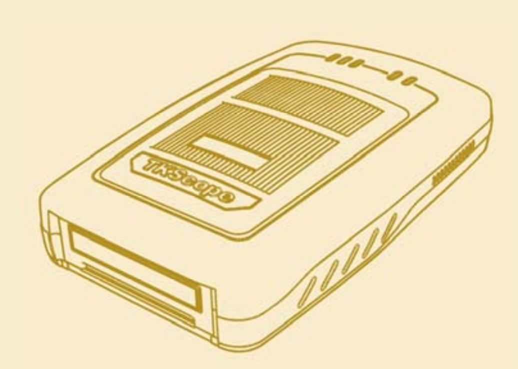
用户在使用仿真器过程中出现工作异常的现象，可以采用硬件自检的方法来排查故障。

如果仿真器硬件自检 100％通过，仿真器肯定是能够正常工作的。

如果硬件自检不能通过，仿真器主机和连接的仿真头组件肯定存在问题，具体问题可以根据自检结果的信息提示来判定。

如仿真器内部器件有损坏，用户选择了错误的 POD 头，仿真电缆接触不良或内部断路，仿真芯片型号错误或损坏，仿真头接触不良等等一系列问题都可以通过硬件自检排查。

TKScope 仿真器的硬件自检功能，解决了用户在使用仿真器过程中，出现莫明其妙的问题时束手无策的尴尬局面。





代码分析和加彩显示功能

4.1 具体实例



TKScope 仿真器具有 512KB 全地址范围内的代码执行覆盖分析和加彩运行轨迹显示的功能，极大的方便了用户分析程序运行的分支流程。

下面将结合具体的实例来讲述如何进行代码执行覆盖分析，如何观察程序运行轨迹加彩显示，让用户体验到直观的效果。程序清单 4.1 为具体实例。

程序清单 4.1　代码执行覆盖分析例程

```
#include   <reg52.h>
#include   <intrins.h>
#define    uchar      unsigned char
#define    uint       unsigned int
uchar      temp[10];

void init(void)                                   //初始化程序
{
        Uchar k;
        for(k=0;k<10;k++)     temp[k]=0;
}

void evaluate1(void)                              //赋值函数 1
{
        uchar t,m;
        for(m=0;m<10;m++)
        {
                t=temp[m]+m;
                temp[m]=t+1;
        }
}

void evaluate2(void)                              //赋值函数 2
{
        uchar n;
        for(n=0;n<10;n++)     temp[n]++;
}

void delay(void)                                   //延时程序
{
        uchar x,y;
        for(x=0;x<100;x++)
                for(y=0;y<200;y++) ;
}

void main(void)
{
        uint i,j;
        init();
        while(1)
        {
                for(i=0;i<6000;i++)
                {
                        evaluate1();              //执行赋值函数 evaluate1
                        delay();                  //延时
                }
                for(j=0;j<6000;j++)
                {
                        evaluate2();              //执行赋值函数 evaluate2
                        delay();                  //延时
                }
        }
}
```

4.2 代码分析

代码执行覆盖分析是指，系统把程序中的各个功能函数的执行情况，以百分比的形式显示出来，便于用户直观的观察到代码的执行覆盖情况。

1 TKScope 仿真器正确设置之后，点击【Debug】进入调试仿真状态，此时可以打开代码执行覆盖分析窗口。选择【View】菜单下的【Code Coverage Window】选项，如图 4.1 所示。

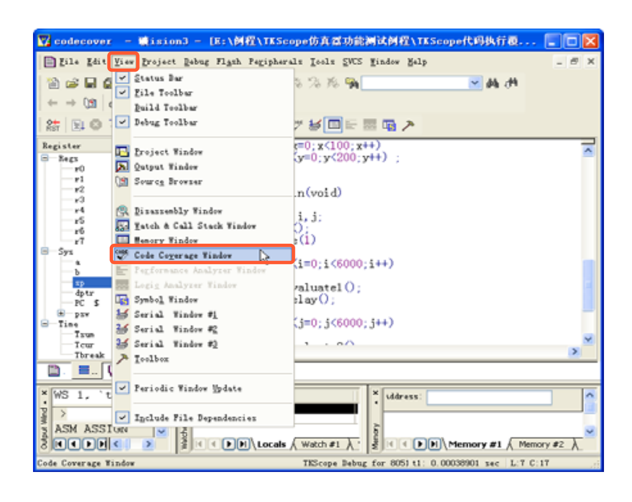


图 4.1　选择代码执行覆盖分析窗口

2 从图 4.2 代码执行覆盖分析窗口中，可以看到程序中的功能函数系统会自动列举出来，各个函数的执行情况和指令条数在相应的函数名后面有所描述。

如图 4.2 中的 INIT 函数，0％ of 7 instructions，含义是 INIT 函数共 7 条指令，没有执行。

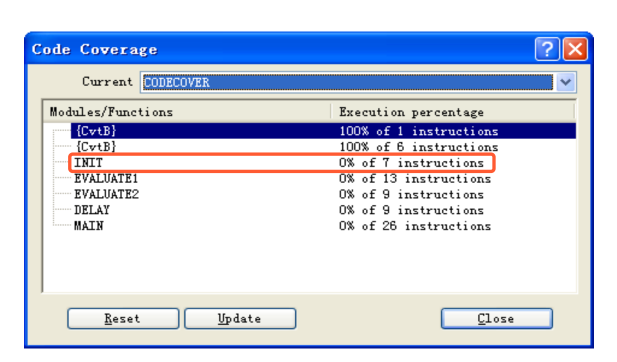


图 4.2　代码执行覆盖分析窗口



这里的指令是指汇编指令，C 函数可通过查看反汇编窗口看到汇编指令。

3 选择【View】菜单下的【Disassembly Window】选项，即可打开反汇编窗口，如图 4.3 所示。

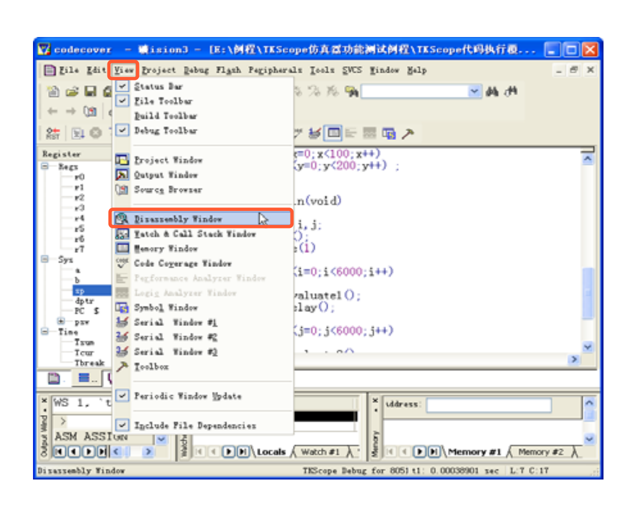


图 4.3　打开反汇编窗口选项

4 在反汇编窗口中可以看到 INIT 函数的全部 7 条汇编指令情况，如图 4.4 所示。

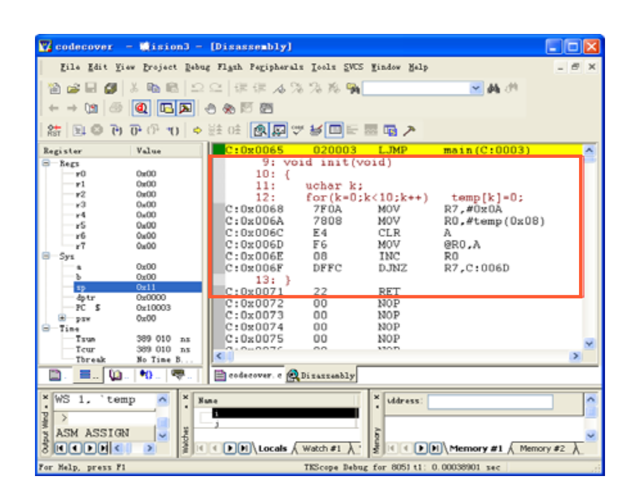


图 4.4　INIT 函数的反汇编程序



其它函数的汇编指令查看方法同 INIT 函数是一样的，用户请自行查看。

5 选择【Debug】菜单下的【Run】选项，全速仿真用户程序，此时可以动态的观察各个函数的执行情况。图 4.5 是在程序仿真运行过程中的截图，从图中可以直观、明了的看到程序中各个功能函数的执行情况。

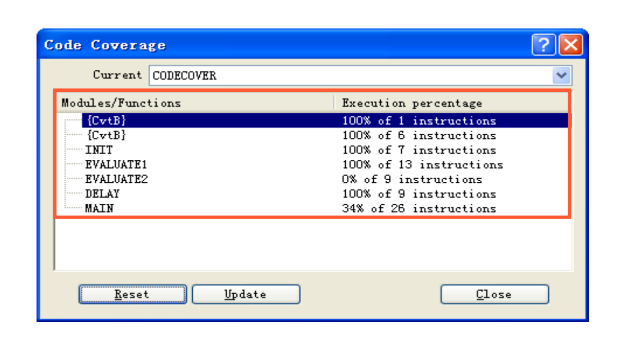


图 4.5　代码执行覆盖分析的结果



图 4.5 中的状态是 MAIN 函数执行了 34%，EVALUATE2 函数还没有执行到，其它的函数已执行完毕。



注意！观察代码执行覆盖分析窗口时，需要将刷新速度调整到最快，以保证显示结果信息的及时性。

6 选择【Peripherals】菜单下的【Running Update】选项，即可打开运行速度刷新窗口，如图 4.6 所示。

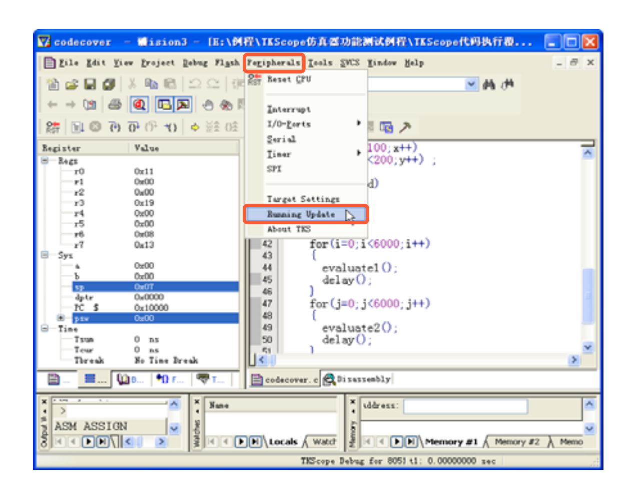


图 4.6　打开运行速度刷新窗口

7 运行速度刷新调整窗口如图 4.7 所示，窗口中显示的结果就是各项刷新速度调整到最快的状态。

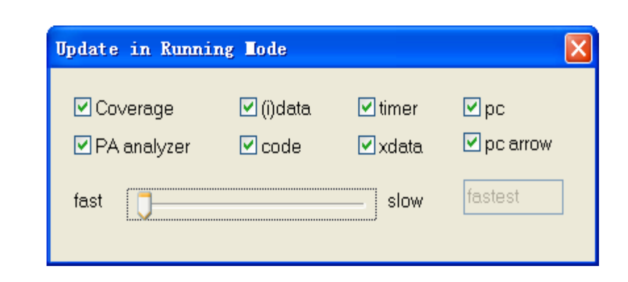


图 4.7　运行速度刷新窗口



观察代码执行覆盖分析时，需要调整到速度刷新最快的状态，以保证显示信息的及时性，可以动态观察代码的执行情况。如果刷新速度很慢，可能会造成显示信息不变、无法动态观察的情况。

4.3 加彩显示



加彩运行轨迹显示是指，系统把执行过的程序指令用彩色标注出来。
在 Keil IDE 开发环境下是在程序行的前端以绿色标注，用户很清楚的看到哪些程序执行过，哪些程序还没有执行。

观察代码执行覆盖分析窗口，在程序还没有执行到 EVALUATE2 函数时，选择【**Debug**】菜单下的【**Stop Running**】选项，让仿真停止下来。

如图 4.8 所示就是加彩运行轨迹显示的结果，用户可以清晰的看到函数的执行情况。此时，MAIN 函数中的第二个 for 循环语句还没有执行，停止时 PC 指针指到 DELAY 函数中，说明程序还没有执行完毕 MAIN 函数中的第一个 for 循环语句。

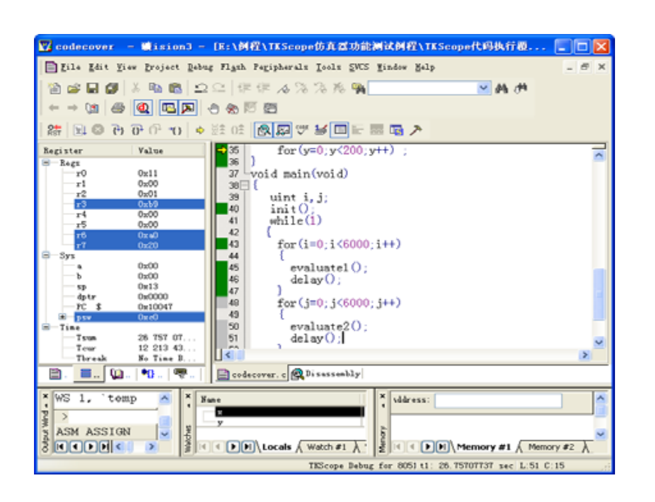


图 4.8　加彩运行轨迹显示

第 5 章



断点操作技巧



5.1 程序运行断点的操作



开始之前！

- 断点是仿真器非常重要的功能，用户在仿真程序过程中几乎离不开断点。用户通过操作断点可以控制仿真器在指定的位置停止运行，然后分析当前的运行状态，判断程序中可能存在的问题或调试整个系统的硬件。
- 断点的种类很多，大体分为简单断点和复杂断点两种。不同的仿真器断点种类也不同，一般都支持简单的程序断点，也是用户经常使用的断点。高档仿真器支持的断点种类很多，如数据读/写断点、代码读取断点、组合断点等。



TKScope 高档专业通用型仿真器支持更多的断点种类，供用户根据需要选择不同的断点来调试程序。

- 64K 程序运行断点：程序运行到该位置时停止运行。
- 64K 代码读取断点：MOVC 指令读取该地址数据时，程序在完成操作后停止运行。
- 64K 数据读取断点：MOVX 指令读取该地址数据时，程序在完成操作后停止运行。
- 64K 数据写入断点：MOVX 指令写入该地址数据时，程序在完成操作后停止运行。
- Tbreak 时间断点：程序运行时间与设置的 Tbreak 时间一致时，程序停止运行。

用户可以单独使用上述断点，也可以混合使用。其中，程序运行断点是用户最频繁使用的断点；Tbreak 时间断点是 TKScope 仿真器独有的断点。用户在使用 TKScope 仿真器时经常使用的就是这两种断点类型，其它 3 种断点属于复杂断点的操作，下面将详细讲解上述 5 种断点的操作技巧。



断点操作包括设置断点、删除断点和关闭断点，其中删除断点和关闭断点对外表现的仿真效果是一样的，但本质是不同的。删除断点是把断点取消，需要时要再次设置；关闭断点是把断点暂时关闭，需要时简单的启动即可。

程序运行断点操作

1 使用鼠标操作设置、删除断点

在程序窗口（包括 C 语言、汇编和反汇编窗口）中，用鼠标双击需要设置断点的程序行，则在窗口左边的状态条中出现红色的断点标志，如图 5.1 所示，完成设置断点操作。

再次用鼠标双击该程序行，窗口左边的红色断点标志消失，则为删除断点操作。

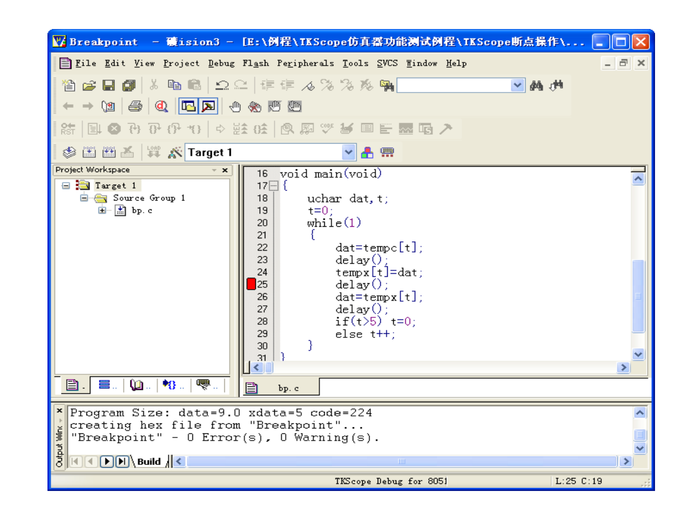


图 5.1　断点标志

2 使用断点菜单或断点工具条操作设置、删除断点

在程序窗口（包括 C 语言、汇编和反汇编窗口）中，使用鼠标或键盘的上移/下移键将光标移动到需要设置断点的程序行，点击快捷图标 或选择【Debug】菜单下的【Insert/Remove Breakpoint】选项，来设置/删除断点。

该程序行没有设置断点，则为设置断点操作；该程序行已经设置断点，则为删除断点操作。

3 关闭断点

对于已经定义的断点，用户如果暂时不用，可以删除断点，也可以关闭断点。点击快捷图标 或选择【Debug】菜单下的【Enable/Disable Breakpoint】选项，来启动/关闭断点。

4 删除所有断点

用户可以点击快捷图标 或选择【Debug】菜单下的【Kill All Breakpoints】选项，来删除程序中设置的所有断点。

5 关闭所有断点

用户可以点击快捷图标 或选择【Debug】菜单下的【Disable All Breakpoints】选项，来关闭程序中设置的所有断点。

5.2 时间断点 Tbreak 的操作

顾名思义，时间断点就是指当程序运行的时间与 Tbreak 设置的时间一致时，程序停止运行。



注意！时间断点 Tbreak 是 TKScope 仿真器独有的断点种类！

仿真器进入监控状态下，在主界面左侧的【**Register**】窗口，可以进行 Tbreak 时间的设置，Tbreak 的时间单位是 ns。

例如，用户需要观察程序运行 6s 时的状态，只需在【**Tbreak**】选项里输入 6 000 000 000，然后按回车即可，如图 5.2 所示。

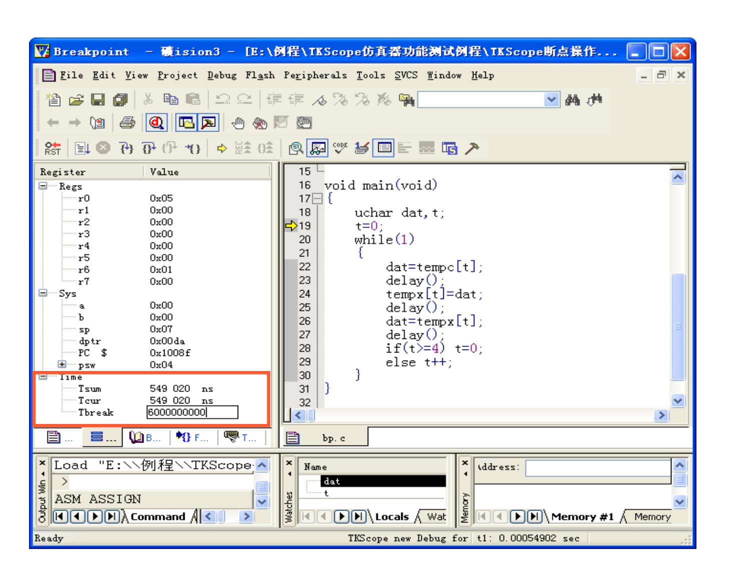


图 5.2　设置时间断点

用户如果要取消时间断点，只需在【**Tbreak**】选项里输入 0，然后按回车即可。此时，【**Tbreak**】选项里显示【**No Time Break**】，表示没有时间断点。

5.3 复杂断点的操作

程序运行断点和时间断点的操作方法很简单，其它三种断点属于复杂断点的操作，用户需要启动断点管理器来进行断点的操作。

断点管理器

选择【**Debug**】菜单下的【**Breakpoints**】选项，即可打开断点管理器，如图 5.3 所示。

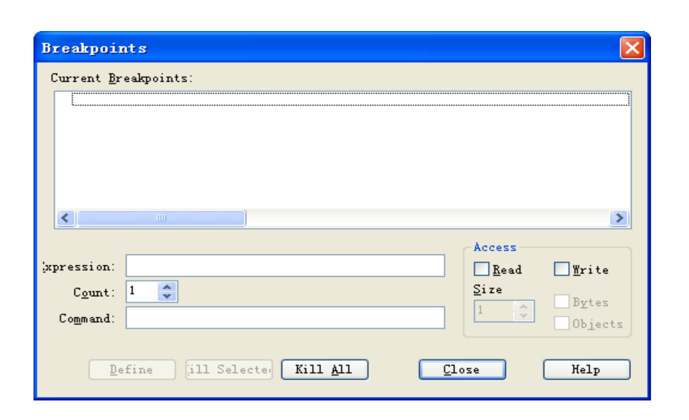


图 5.3　断点管理器

断点管理器窗口的主要功能模块含义如下。

功能模块	意义
Current Breakpoints	当前所有断点的显示窗口。
Expression	断点表达式。
Count	触发断点的次数（TKScope 不支持）。
Command	断点命令输入（TKScope 不支持）。
Read	断点操作属性为读。
Write	断点操作属性为写。
Size	断点的尺寸以 Byte 为单位，TKScope 支持 8bits 尺寸（Size＝1）。
Byte	TKScope 不支持选择。
Objects	TKScope 不支持选择。
Define	定义断点。
Kill Selectd	删除选择的断点。
Kill All	删除全部的断点。

具体实例

复杂断点的操作涉及到程序中具体的函数名、变量名等，所以需要结合下面实例来讲解，程序清单 5.1 为具体的实例。

程序清单 5.1　复杂断点例程

```
#include<reg52.h>
#include<intrins.h>
#define uchar unsigned char
code char tempc[5]={0x00,0x01,0x02,0x03,0x04};
xdata char tempx[5]={0x00};

void delay(void)                                    //延时子程序
{
        uchar x,y;
        for(x=0;x<100;x++)
                for(y=0;y<200;y++) ;
}

void main(void)
{

        uchar dat,t;
        t=0;
        while(1)
        {
                dat=tempc[t];                       //读代码段数据，执行 MOVC 指令
                delay();
                tempx[t]=dat;                       //写数据到 xdata 空间，执行 MOVX 指令
                delay();
                dat=tempx[t];                       //读 xdata 空间数据，执行 MOVX 指令
                delay();
                if(t>=4) t=0;
                else t++;
        }
}
```

程序运行断点可以采用前面介绍的简单方法操作，也可以在断点管理器中进行操作。在断点管理器中的操作有两种方法。

1 在【Expression】中写入函数名称。

例如，在上面实例中的 main 函数入口处设置断点，只需在【**Expression**】中写入“**main**”，【**Access**】中的【**Read**】和【**Write**】属性均不选择，如图 5.4 所示，然后点击【**Define**】即可。

此时，在【**Current Breakpoints**】窗口可以看到定义的断点列表，在 C 程序窗口和反汇编窗口对应的程序行可以看到断点的红色标志。

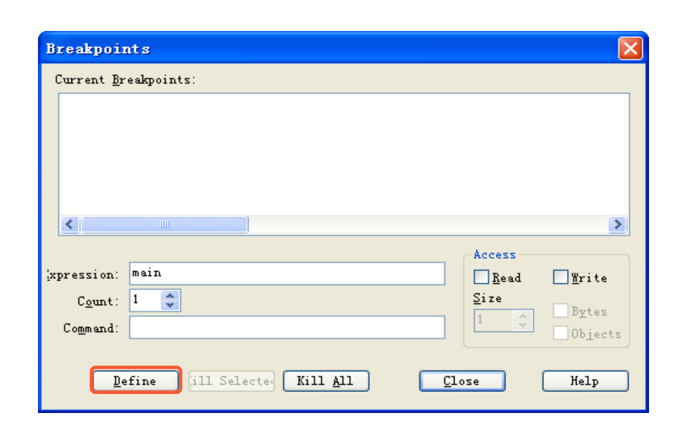


图 5.4　定义程序运行断点



使用函数名称定义断点的方法，用户无需知道断点的具体地址，只需输入函数名称，方便用户记忆。即使程序重新编译，函数地址发生变化，系统能够自动重新调整断点的位置。

2 在【Expression】中写入函数地址。

例如，同样的 main 函数入口处设置断点，此时需要知道 main 函数的具体地址，通过查看反汇编窗口可以获得地址，如图 5.5 所示。在【**Expression**】中写入“**C:0x008F**”，其它设置同方法一。其中，C 表示是程序属性，0x008F 是 main 函数的地址（不同的编译环境，函数地址值可能不同）。

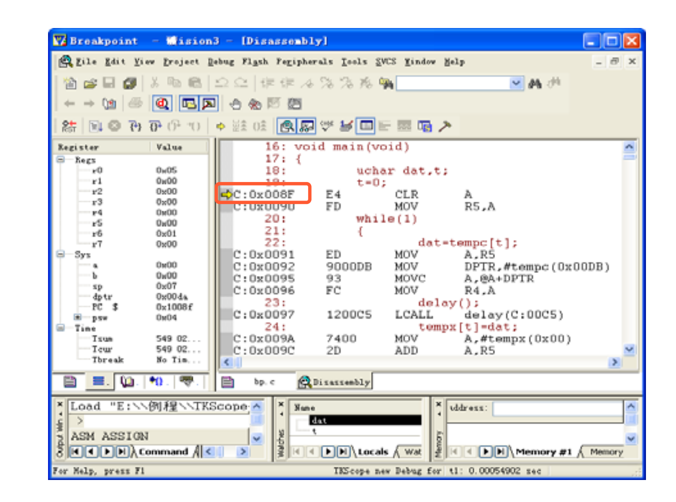


图 5.5　查看 main 函数的具体地址



使用函数地址定义断点的方法，用户必须知道函数的准确地址。

定义 MOVC 代码读取断点

MOVC 代码读取断点操作属于复杂断点操作，只能在断点管理器中进行。在断点管理器中的操作有两种方法。

1 **在【Expression】中写入代码变量名称。**

例如，上面实例中定义了“code char tempc[5]”这样一个数组，用户如果要查看读取 tempc[3]后的程序运行状态，可以在 tempc[3]位置添加 MOVC 读取断点。

在【**Expression**】中写入 tempc[3]，【**Access**】中的【**Read**】属性选中【**Write**】属性不选中，如图 5.6 所示，然后点击【**Define**】即可。

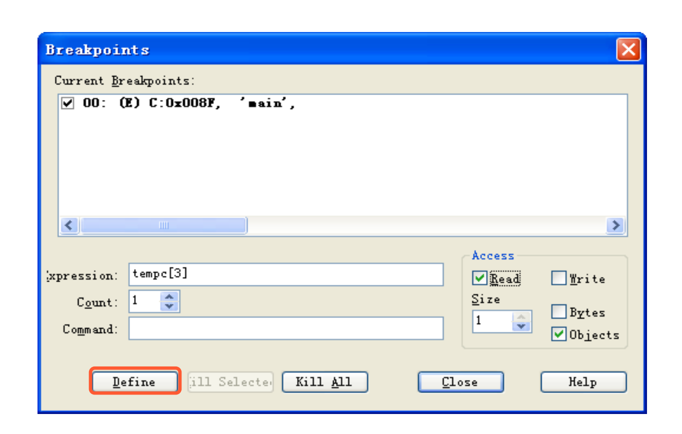


图 5.6　定义 MOVC 读取断点

2 **在【Expression】中写入代码变量地址。**

例如，同样的在 tempc[3]位置添加 MOVC 读取断点，此时需要知道 tempc[3]的具体地址，通过查看反汇编窗口可以获得地址，如图 5.7 所示。Tempc 的地址是 0x00DB，则 tempc[3]的地址是 0x00DE。在【**Expression**】中写入“C: 0x00DE”其它设置同方法一。

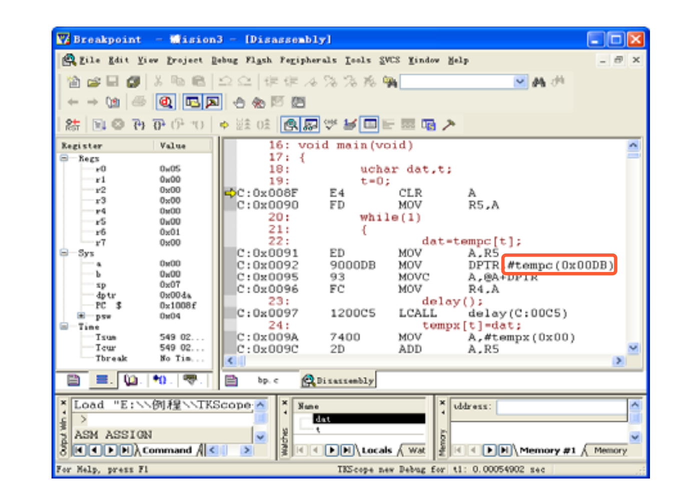


图 5.7　查看 tempc 变量的具体地址



使用函数地址定义断点的方法，用户必须知道函数的准确地址。



注意！在定义 MOVC 读取断点时，【**Read**】属性一定要选中，否则将成为程序运行断点。【**Write**】属性不能选中，因为程序代码无法通过指令进行写操作。

MOVX 数据读取断点操作属于复杂断点操作，只能在断点管理器中进行。在断点管理器中的操作有两种方法。

1 在【Expression】中写入外部数据变量名称。

例如，上面实例中定义了“xdata char tempx[5]”这样一个外部数据空间数组，用户如果要查看读取 tempx[2]后的程序运行状态，可以在 tempx[2]位置添加 MOVX 读取断点。

在【Expression】中写入 tempx[2]，【Access】中的【Read】属性选中【Write】属性不选中，如图 5.8 所示，然后点击【Define】即可。

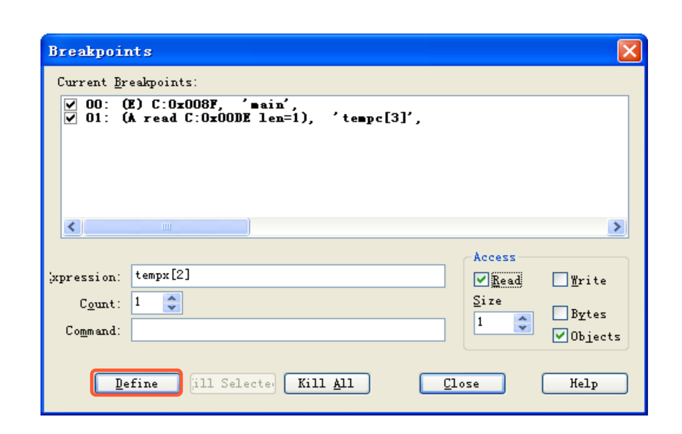


图 5.8　定义 MOVX 读取断点

2 在【Expression】中写入外部数据变量地址。

例如，同样的在 tempx[2]位置添加 MOVX 读取断点，此时需要知道 tempx[2]的具体地址，通过查看反汇编窗口可以获得地址，如图 5.9 所示。Tempx 的地址是 0x00，则 tempx[2]的地址是 0x02。在【Expression】中写入“X: 0x000002”其它设置同方法一。其中，X 表示是外部数据空间属性，0x000002 是 tempx[2]的数据地址。

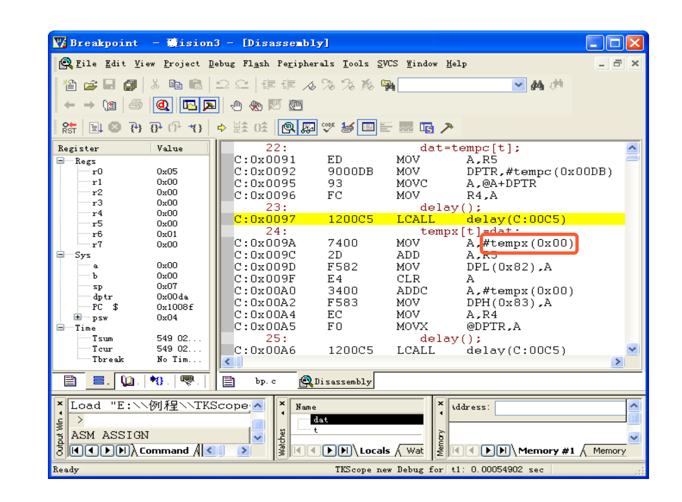


图 5.9　查看 tempx 变量的具体地址



使用外部数据变量地址定义断点的方法，用户必须知道数据变量的准确地址。

MOVX 数据写入断点操作属于复杂断点操作，只能在断点管理器中进行。在断点管理器中的操作有两种方法。

1 **在【Expression】中写入外部数据变量名称。**

例如，上面实例中定义了“xdata char tempx[5]”这样一个外部数据空间数组，用户如果要查看写入数据到 tempx[4]后的程序运行状态，可以在 tempx[4]位置添加 MOVX 写入断点。

在【**Expression**】中写入 tempx[4]，【**Access**】中的【**Write**】属性选中【**Read**】属性不选中，如图 5.10 所示，然后点击【**Define**】即可。

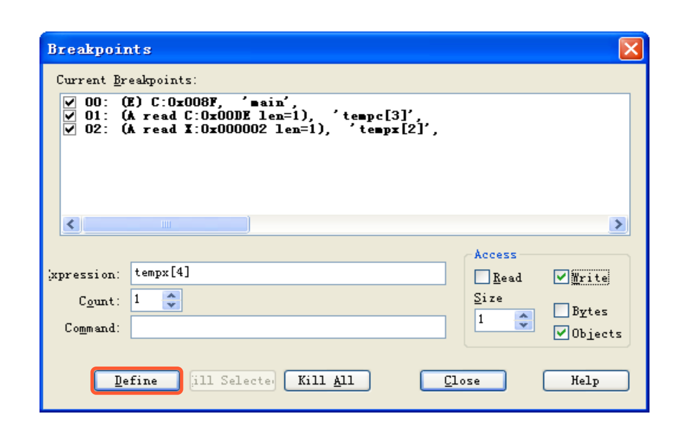


图 5.10　定义 MOVX 写入断点

2 **在【Expression】中写入外部数据变量地址。**

例如，同样的在 tempx[4]位置添加 MOVX 写入断点，此时需要知道 tempx[4]的具体地址，通过查看反汇编窗口可以获得地址，如图 5.9 所示。Tempx 的地址是 0x00，则 tempx[4]的地址是 0x04。在【**Expression**】中写入“X: 0x000004”其它设置同方法一。



使用外部数据变量地址定义断点的方法，用户必须知道数据变量的准确地址。

注意！ 在定义 MOVX 读取/写入断点时，【**Read**】/【**Write**】至少必须选中一项，否则将不能形成有效的 MOVX 操作断点。如果【**Read**】/【**Write**】全部选中，则该断点在读取/写入任何操作发生时都会形成有效的断点。

运行程序观察现象

设置完上述断点后，全速运行程序，在反汇编窗口中，可以看到程序分别运行到如下位置时停止运行。

Main 函数入口处	反汇编指令　C：0x008F　E4　CLR　A
MOVC 读取 tempc[3]后	反汇编指令　C：0x0095　93　MOVC　A，@A+DPTR
MOVX 读取 tempx[2]后	反汇编指令　C：0x00B3　E0　MOVX　A，@DPTR
MOVX 写入 tempx[4]后	反汇编指令　C：0x00A5　F0　MOVX　@DPTR，A

对于【**Current Breakpoints**】窗口中存在的各个断点，用户可以暂时关闭，使其断点作用失效。

方法是点击断点表达式前面的打勾框，使之取消打勾。由于失效的断点仍然存在于【**Current Breakpoints**】窗口中，因此用户可以在需要的时候重新启用，这样比删除断点然后再添加断点方便得多。如图 5.11 所示，就是关闭 main 函数入口处设置的断点。

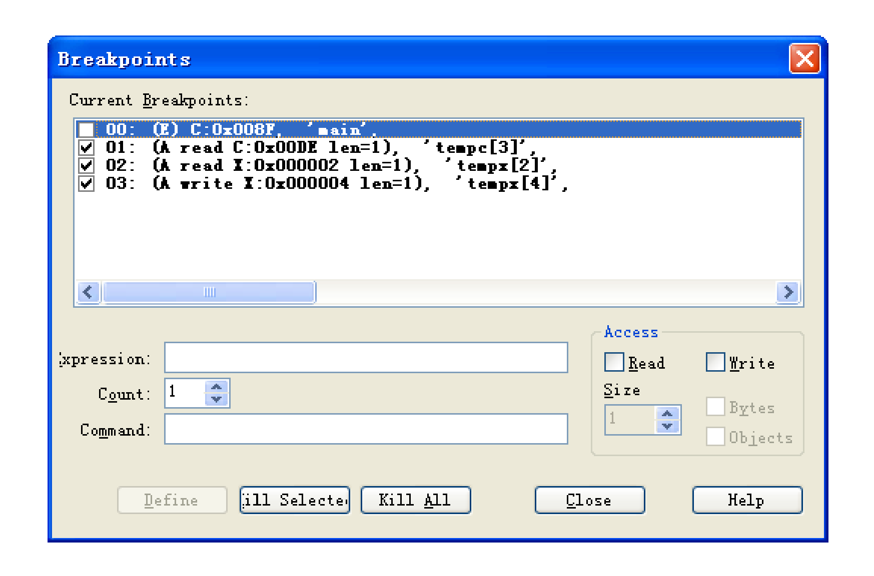


图 5.11　关闭 main 函数处的断点




第 6 章
逻辑功能使用详解

6.1 性能分析器的使用方法



性能分析（Performance Analyzer）是用来分析代码和数据的执行效率，并以统计图形的方式表现出来。
TKScope 仿真器内置的逻辑板可以同时完成代码和数据的性能分析。根据硬件的配置，最大可以同时进行 512KB 代码和 512KB 数据的性能分析。

性能分析器是 TKScope 仿真器的一项功能强大的仿真分析功能，对于用户分析程序非常有帮助。下面将结合实例，来详细讲解使用方法。

6.1.1 具体实例及功能设置

性能分析器涉及到具体函数名和变量等的设置，为了更加清晰的描述，将结合下面的实例来详细的讲述具体的使用方法。程序清单 6.1 为具体的实例。

程序清单 6.1　性能分析器功能例程

```
#include<reg52.h>
#include<intrins.h>
#define uchar unsigned char
#define uint unsigned int

void patest1(void)
{
        uchar t1=0;
}

void patest2(void)
{
        uchar t2=0;
}

void main(void)
{
        uint x,y;
        while(1)
        {
                for(x=0;x<3000;x++)        patest1();
                for(y=0;y<6000;y++)        patest2();
        }
}
```

TKScope 仿真器的设置方法在《**第 2 章—仿真环境设置**》已经详细描述，这里不再重复叙述。需要重点强调的是，使用性能分析器时在【**逻辑功能**】设置选项中，必须选择【**性能分析器**】这项，如图 6.1 所示。

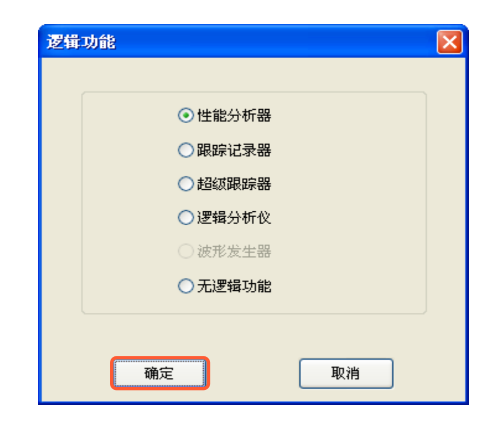


图 6.1　性能分析器设置窗口

6.1.2 性能分析定义

TKScope 仿真器正确设置之后，点击【**Debug**】进入调试仿真状态，此时选择【**Debug**】菜单下的【**Performance Analyzer**】选项，如图 6.2 所示，即可弹出性能分析定义窗口，如图 6.3 所示。

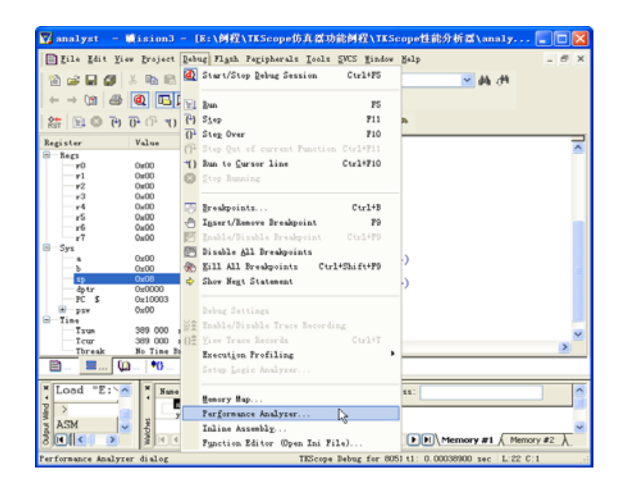


图 6.2　选择性能定义选项

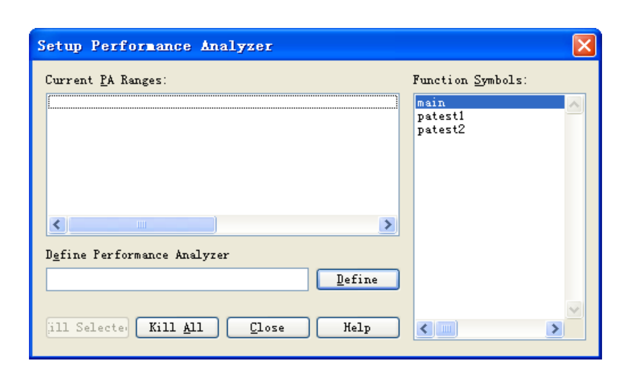


图 6.3　性能分析定义窗口

1 功能函数的定义

在图 6.3 中可以看到，右侧的【**Function Symbols**】对话框中，系统自动检索列举出程序中涉及到的所有的功能函数。用户根据实际观察的需要选择定义相应的函数，双击函数名，在【**Define Performance Analyzer**】对话框中出现此函数名，如图 6.4 所示，然后点击右侧的【**Define**】即可在【**Currrent PA Ranges**】窗口中看到定义的功能函数，如图 6.5 所示。

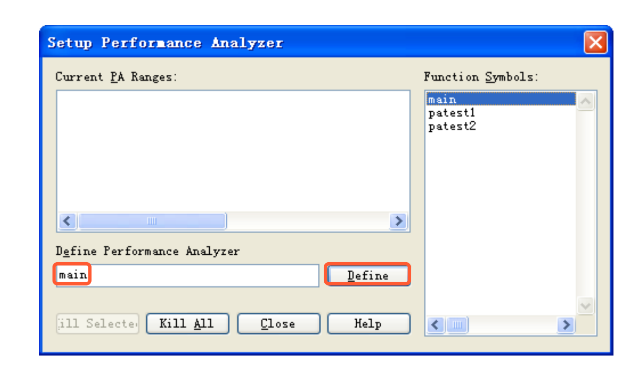


图 6.4　定义功能函数 main

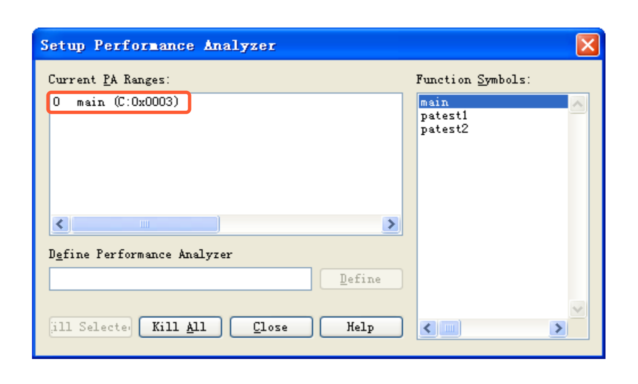


图 6.5　功能函数 main 定义结果

2 变量的定义

对于用户程序里面定义的各种变量，包括特殊功能寄存器，用户可以这样定义。

在【**Define Performance Analyzer**】对话框中输入“**&+变量名**”，然后，点击右侧的【**Define**】即可。如例程中的变量 x、y 可以采用这种方法定义，如图 6.6 所示。

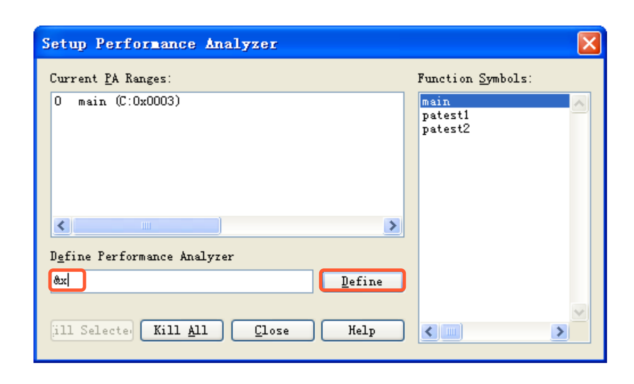


图 6.6 定义变量和特殊功能寄存器的方法



对于数组的定义，有两种方法。例如程序中存在数组 temp[10]，一是直接定义“temp”，得到的是一个地址；二是定义“&temp[0], &temp[9]”得到的是一个范围。



注意！对于数组的定义，只能选择其中一种方式定义，两者不能同时定义。

3 代码地址范围的定义

用户可以根据实际观察分析的需要定义一段代码段，如 0x0010～0x0020 这段代码，直接在【**Define Performance Analyzer**】对话框中输入“**0x0010，0x0020**”，然后，点击右侧的【**Define**】即可，如图 6.7 所示。

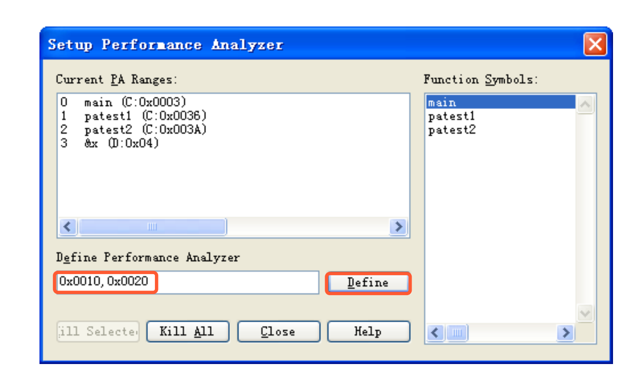


图 6.7 定义代码地址范围的方法

4 取消定义

用户如果要取消某些定义选项，可以选中要取消的定义，然后点击【**Kill Select**】即可删除，如图 6.8 所示。用户如果要取消所有的定义选项，直接点击图 6.8 中的【**Kill All**】即可。

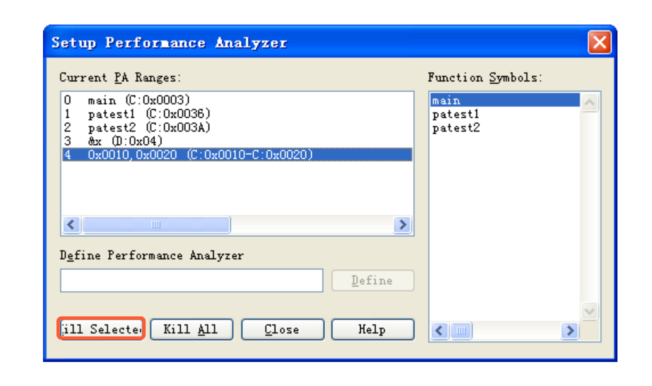


图 6.8 取消定义方法

6.1.3 性能分析实施

1 性能分析定义完毕之后，点击【**Close**】关闭性能分析定义窗口，然后选择【**View**】菜单下的【**Performance Analyzer Window**】选项，如图 6.9 所示，即可打开性能分析显示窗口，如图 6.10 所示。

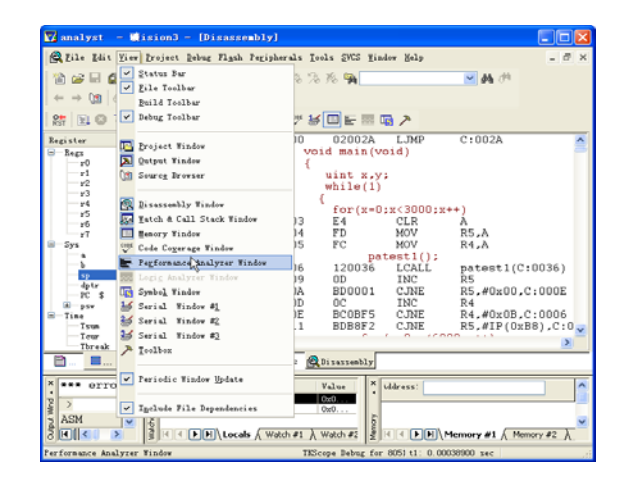


图 6.9　打开性能分析窗口

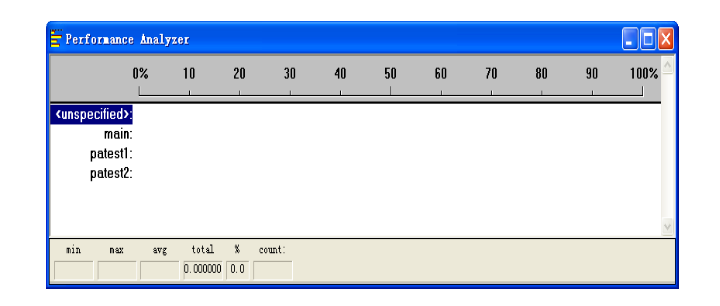


图 6.10　性能分析观察窗口

2 选择【**Debug**】菜单下的【**Run**】选项，全速仿真用户程序，此时，可以动态的观察各个定义选项的运行情况。当然，也可以选择【**Debug**】菜单下的【**Stop Running**】选项，让仿真停止下来再进行观察分析。

程序仿真运行之后，性能分析窗口会显示各个定义选项的执行情况，如图 6.11 所示。

点击具体的某一个选项，则该项的一些运行记录数据会在下面的提示框中显示。例如点击 main 函数，提示框中会显示出 main 函数的一些相关数据，如图 6.12 所示。

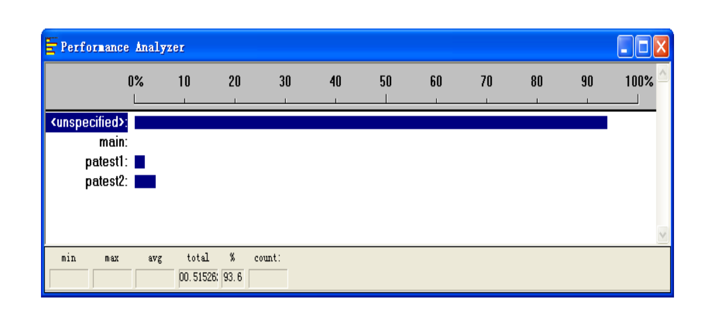


图 6.11　性能分析结果显示窗口

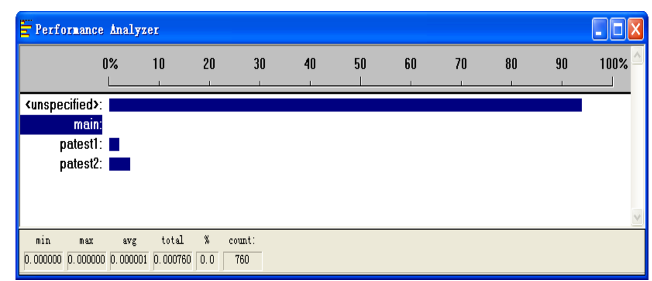


图 6.12　功能函数 main 的数据显示

下面介绍各个数据提示框在硬件仿真时的意义。对于功能函数和程序地址范围来说，各项的意义请详见表 6.1；对于空间变量来说，各项的意义请详见表 6.2。

表 6.1　功能函数和程序地址范围的参数定义

名称	意义
min	无意义。
max	无意义。
avg	在选择的地址范围或函数内消耗的总时间。
total	程序同等操作消耗的时间总数。
%	avg/total 的百分比。
count	在选择的地址范围或函数上进入其实地址的次数。

表 6.2　空间变量的参数定义

名称	意义
min	无意义。
max	无意义。
avg	在选择的变量地址范围内所有地址被操作的总次数。
total	程序同等操作的总次数。
%	avg/total 的百分比。
count	变量字节起始地址被操作的次数（有可能和其它字节地址操作次数不同）。

6.2 超级跟踪功能的妙用



在英文里，Trace 的意思是跟踪；在仿真调试环境里，Trace 的意思是跟踪已经被执行过的程序的运行轨迹。
Trace 功能是仿真器中使用的先进技术，主要用于查看程序运行中各种状态的连续变化，用来实时记录运行的程序轨迹，便于用户分析程序的运行流程。

TKScope 仿真器具有超级跟踪记录功能（**Super Trace**）和跟踪记录功能（**History Tarce**）。

TKS-B 系列仿真器具有 64KB 容量的 Trace 功能（TKS-52B 除外），使用过 B 系列仿真器的用户，应该会感受到 Trace 功能的实用性和方便性。

TKScope 系列仿真器在 Trace 功能的基础上又增加了时间记录功能，也就是说 TKScope 仿真器在记录程序运行轨迹的同时还以 32 位容量记录每步程序运行的消耗时间。

TKScope 仿真器具有 **512KB** 的超大容量记录空间，用户根据程序运行轨迹和时间消耗轨迹，可以更完整的分析程序运行。此外，TKScope 仿真器还具有超级跟踪记录功能，**Super Trace** 为独创的增强性跟踪记录，除保留了 History Trace 的全部功能外，还可实现跟踪记录芯片内部的主要资源，如 ACC / B / DPTR / SP / R0-7 等（根据不同的芯片和硬件可记录的资源情况可能不同），极大提高了跟踪记录的实用性。



Super Trace 超级跟踪，全球首例最新独创专利技术。

- 全球首个 8051 全范围 SFR 跟踪记录，更好的协助用户分析程序运行轨迹。
- 最大 512KB 超大容量实时 Super Trace 超级跟踪功能。
- 跟踪记录 ACC / B / DPTR / SP 等全部 SFR。
- 跟踪记录内部 4 组 R0-R7 寄存器。
- 同时记录 48-bit 同步时间记录标签（Time Stamp），10ns 精度。
- 同时记录程序地址指针 PC。



Trace 功能是连续记录程序的运行状态，需要一个高速缓冲区进行实时数据记录，无论这个高速缓冲区容量有多大，在程序实时运行后都将会溢出。因此，仿真器中的 Trace 功能是采用向后记录的方式，溢出后最前面的数据（旧数据）将被新数据覆盖，程序运行中的 Trace 缓冲区中总是记录程序的最新状态数据。当程序停止运行后，用户可以查看缓冲区中的状态数据，以此来分析排除程序故障。

下面将结合具体的实例来说明超级跟踪记录功能（Super Trace）和跟踪记录功能（History Tarce）的使用方法和功能。

6.2.1 具体实例及功能设置

由于超级跟踪记录功能和跟踪记录功能很相近，在使用方法上两者是一样的，只是在观察结果时略有不同，所以下面按照超级跟踪记录功能来讲解具体的使用方法，在观察结果时把两种功能的观察结果做下比较分析。程序清单 6.2 为具体的实例。

程序清单 6.2　超级跟踪记录功能例程

```
#include<reg52.h>
#include<intrins.h>
#define uchar unsigned char
uchar t1,t2,t3;

void evaluate (void)                                    //t1、t2、t3 赋值函数
{
        t1=1;
        t2=0;
        t3=0;
}

void main(void)
{
        evaluate();
        while(1)
        {
                if(t1)                                  //t1＝1 时，执行 t1 减 1，t2、t3 加 1 操作
                {
                        t1--;
                        t2++;
                        t3++;
                }
                else evaluate();                        //t1＝0 时，执行 t1、t2、t3 赋值函数
        }
}
```

TKScope 仿真器的设置方法在《**第 2 章—仿真环境设置**》已经详细描述，这里不再重复叙述。

需要重点强调的是，使用超级跟踪记录功能时，在【**逻辑功能**】设置选项中，选择【**超级跟踪器**】这项，如图 6.13 所示；使用跟踪记录功能时，选择【**跟踪记录器**】这项，如图 6.14 所示。

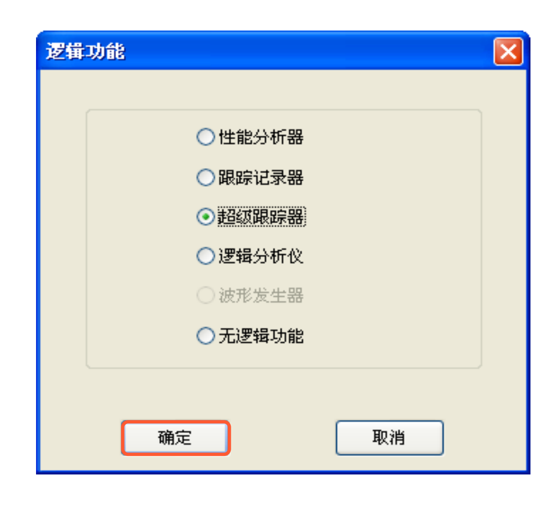


图 6.13　超级跟踪器设置窗口

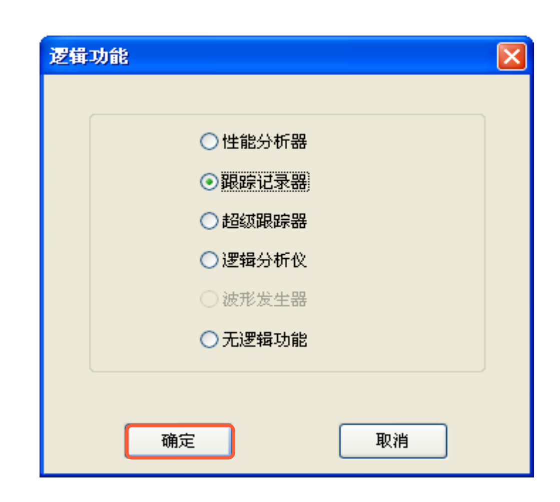


图 6.14　跟踪记录器设置窗口

6.2.2 功能实现

1 TKScope 仿真器正确设置之后，点击【**Debug**】进入调试仿真状态。如果界面如图 6.15 所示，红色方框中的图标是亮色的，则表示已经打开跟踪记录。如果界面如图 6.16 所示，红色方框中的图标是灰色的，则表示跟踪记录没有打开，此时需要打开跟踪记录。

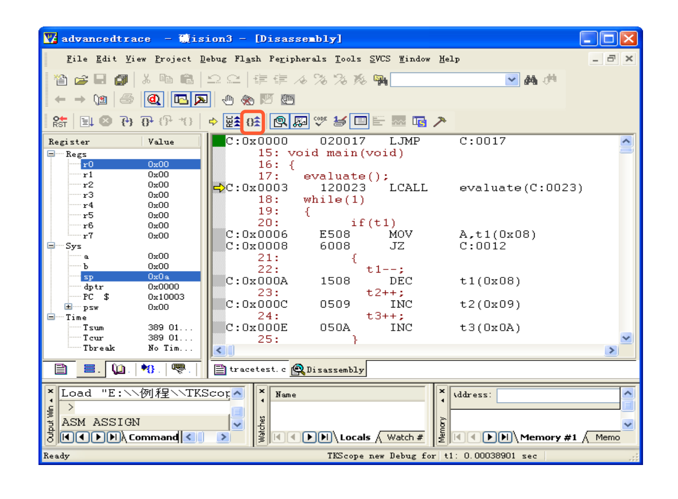


图 6.15　跟踪记录打开的界面

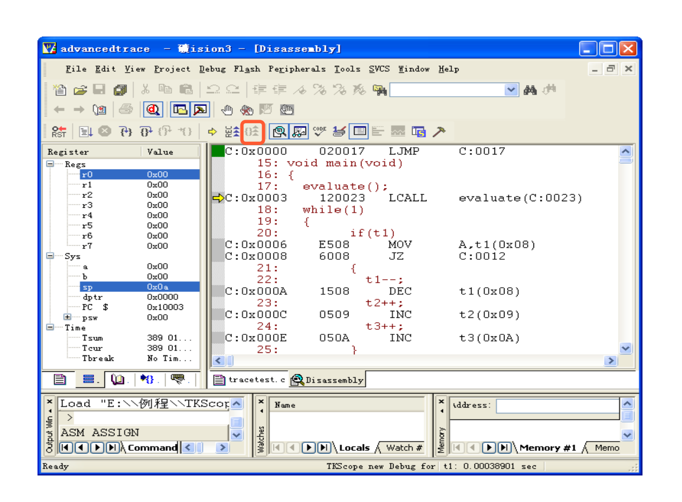


图 6.16　跟踪记录没有打开的界面

2 打开跟踪记录的方法很简单，选择【**Debug**】菜单下的【**Enable/Disable Trace Recording**】选项（如图 6.17 所示），或点击快捷图标 ，即可打开跟踪记录。跟踪记录打开后的界面如图 6.15 所示。

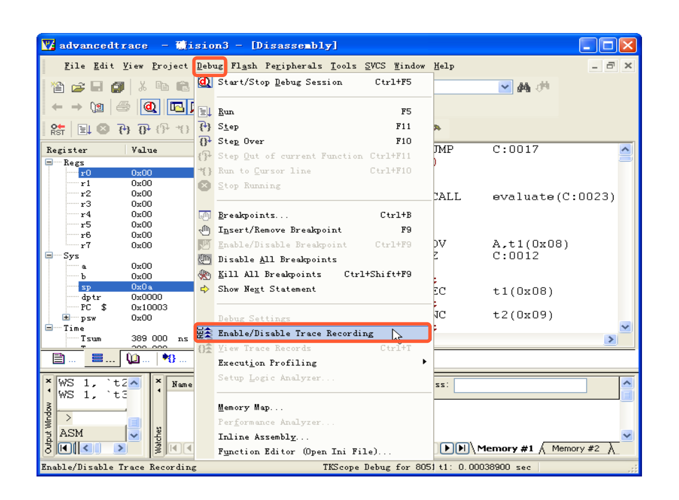


图 6.17　打开跟踪记录选项

6.2.3 结果分析

1 选择【**Debug**】菜单下的【**Run**】选项或点击快捷图标，全速运行程序。当程序遇到断点停止下来或点击快捷图标停止运行，此时可以在反汇编窗口观察到跟踪记录的结果。

2 选择【**Debug**】菜单下的【**View Trace Records**】选项（如图 6.18 所示）或点击快捷图标，即可在反汇编窗口看到跟踪记录的结果，如图 6.19 所示。

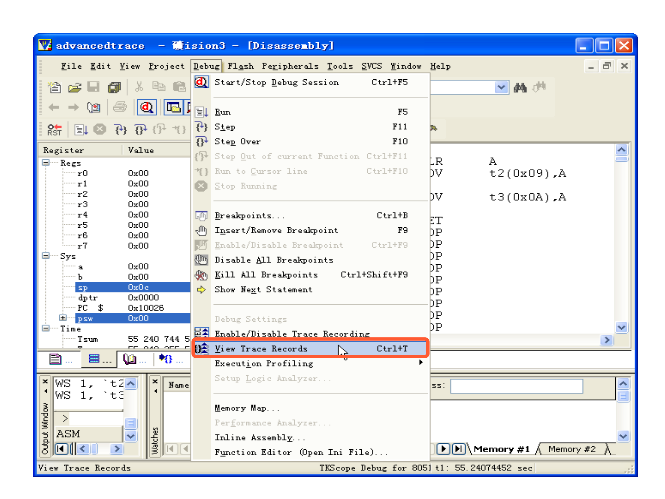


图 6.18　观察跟踪结果查看选项

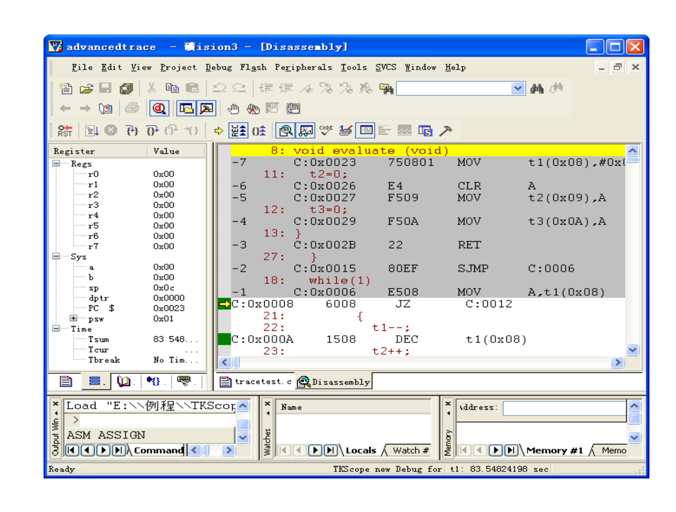


图 6.19　跟踪记录结果查看窗口



在跟踪记录的窗口中，用户可以非常清楚的看出程序运行轨迹。被覆盖的区域中每一个反汇编程序行的前方都有－n（n＝1，2，3…）标记，它表明了程序执行的先后次序。“－1”表明该行是刚执行过的程序行，“－2”表明是在“－1”前执行的那个程序行，依次类推。

由于窗口尺寸所限，当前反汇编窗口只能显示跟踪记录的一小部分。用户可以使用计算机键盘上的【**Page Up/Page Down**】键来变换窗口的显示内容，也可以使用鼠标点击反汇编窗口中右上角/右下角的上移/下移箭头。

! 注意！反汇编窗口右边的滑动条不能用来移动 Trace 窗口的显示内容，否则会退出 Trace 窗口。如果退出了 Trace 窗口，用户需要再次打开窗口。

3 超级跟踪记录的结果

当鼠标点击某行程序时，黄色移动条就指到某行程序，此时在左侧的【**Register**】窗口中可以看到时间的记录情况和主要特殊功能寄存器的记录情况，更新的数据系统用蓝色标注，更加方便用户分析程序。

如图 6.20 所示，程序执行完 RET 指令，即弹栈操作，SP 的值为 0x0a，在【**Register**】窗口中可以看到。

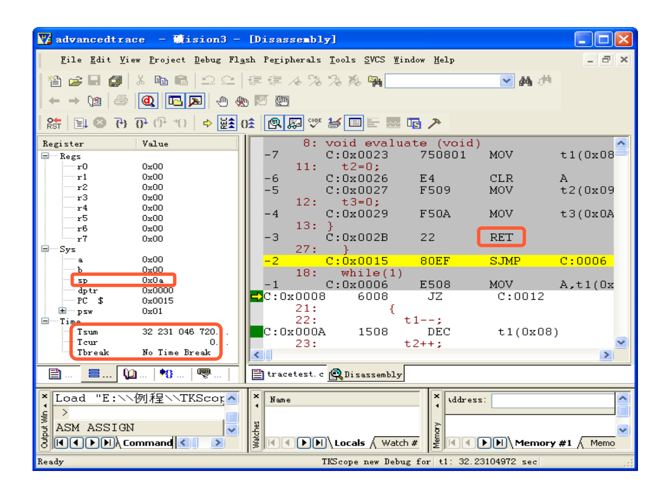


图 6.20　超级跟踪记录的观察结果 1

如图 6.21 所示，程序没有执行 RET 指令前，即弹栈操作还没有发生，SP 的值为 0x0c，在【**Register**】窗口中可以看到。另外，在【**Register**】窗口中还可以看到时间的记录情况。

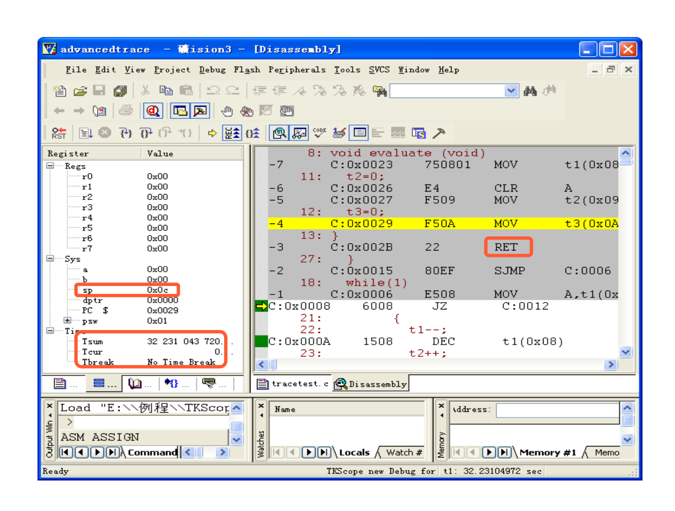


图 6.21　超级跟踪记录的观察结果 2

这些宝贵的提示信息，让用户更加清晰的掌握程序的运行轨迹和指令的执行情况，非常方便用户排查程序错误。

4 跟踪记录器的结果

跟踪记录器的使用方法同上，观察到的结果比跟踪记录器少了一些特殊寄存器的记录。当鼠标点击某行程序时，在左侧的【**Register**】窗口中可以看到时间的记录情况，但不能看到特殊功能寄存器的值。

图 6.22 的状态与图 6.20 的状态相对应。

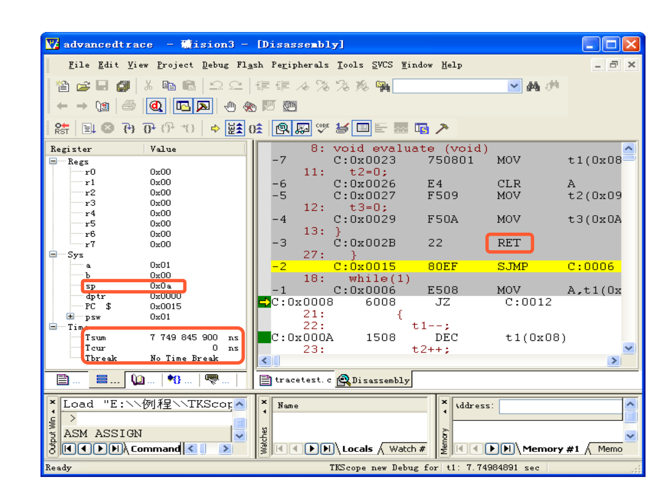


图 6.22　跟踪记录的观察结果 1

图 6.23 的状态与图 6.21 的状态相对应。

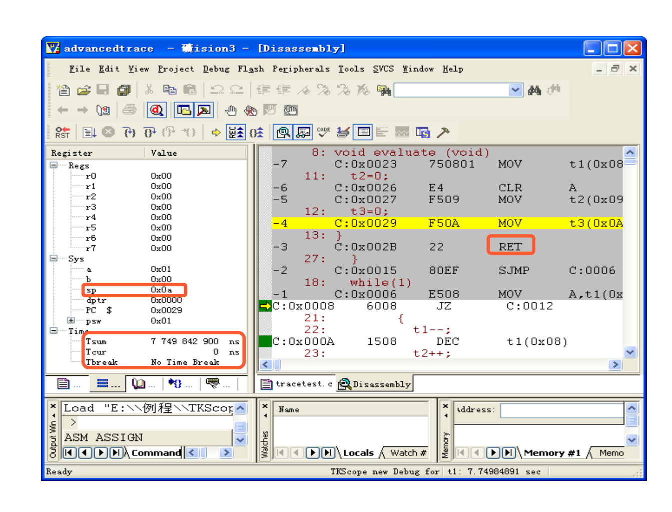


图 6.23　跟踪记录的观察结果 2

图 6.22 和图 6.23 是在跟踪记录器功能下观察到的结果，我们通过比较可以看到 SP 的值没有变化，系统显示的值是在程序停止下来时保存的当前值，但是时间的记录情况可以看出。

跟踪记录器虽然没有超级跟踪器功能强大，但是与其它的仿真器相比，增加了时间的记录功能，已经有了很大的提高，更加方便用户跟踪分析程序。

6.3 逻辑分析仪的使用方法



逻辑分析仪是电子工程师开发过程中的好帮手，越来越广泛的应用到数字电路开发中，其集成逻辑分析、总线分析、协议分析、频率计、逻辑笔等众多功能于一身，让您无需火眼金晶就能轻松发现隐藏在系统中的错误。

TKScope 仿真器内置的 LAB 逻辑板在仿真的同时，还可独立运行最大 64 路的逻辑分析功能。64 路分 32 路内部仿真信号和 32 路外部信号，逻辑分析的启动和停止，可以与仿真器的状态相关联，也可以独立的作为逻辑分析仪来使用。

TKScope 仿真器将仿真和逻辑分析功能结合在一起，让您的程序仿真状态来控制逻辑分析的启动和停止，这样您可以更加方便的排查系统中存在的错误和隐患。



真正专业的内嵌逻辑分析仪，zlglogic 全面支持。

- 最大 **64 路**输入信号，**200M** 采样速度，**1MB** 存储深度。
- 创新的触发测量方式和崭新的分析测量手段，令测量更加简单快捷。
- 可设置边缘/电平/总线等基本触发方式和高级复杂触发方式，方便易用。
- 人性化软件轻松完成信号测量、触发设置、动态帮助、软件升级等功能。
- 内部 32 路缺省仿真 MCU 信号，参与显示和触发。
- 仿真模块和逻辑分析仪模块紧密关联，可同时独立运行或停止。
- 多文档结构可让您在测量的同时观察和比较其它数据。
- 强大的数据导出功能支持对测量信号进行二次分析成为可能。
- 柔性频率设置突破传统的 1、2、5 进制，使得测量更加精确。
- 动态升级的硬件算法使您的测量手段与时并进。

TKScope 仿真器独立的作为逻辑分析仪使用，方法和成品逻辑分析仪相同，这里不做介绍，用户请参考逻辑分析仪的用户使用手册。本篇主要讲解 TKScope 仿真器的逻辑分析功能与仿真状态相关联的使用方法。

6.3.1 具体实例及功能设置

下面将结合具体的实例，来详细的讲解逻辑分析仪功能的使用方法，并展示逻辑分析仪软件界面的显示结果。程序清单 6.3 为具体的实例。

程序清单 6.3　逻辑分析仪功能例程

```
#include<reg52.h>
#include<intrins.h>
#define uchar unsigned char
sfr    AUXR = 0x8E;
xdata char tempx[6]={0x00};
code char tempc[6]={0x55,0xAA,0x55,0xAA,0x55,0xAA};

void main(void)
{
        uchar dat,i,j;
        while(1)
        {
                AUXR=0x01;                          //对内部扩展 XDATA 操作
                for(i=0;i<6;i++)
                {
                        dat=tempc[i];
                        tempx[i]=dat;               //写内部扩展数据空间，WR 无效
                        dat=tempx[i];               //读内部扩展数据空间，RD 无效
                }
                AUXR=0x03;                          //对外部 XDATA 操作
                for(j=0;j<6;j++)
                {
                        dat=tempc[j];
                        tempx[j]=dat;               //写外部数据空间，WR 有效
                        dat=tempx[j];               //读外部数据空间，RD 有效
                }
        }
}
```

TKScope 仿真器的设置方法在《**第 2 章—仿真环境设置**》已经详细描述，这里不再重复叙述。需要重点强调的是，使用逻辑分析仪时，在【**逻辑功能**】设置选项中，必须选择【**逻辑分析仪**】这项，如图 6.24 所示。

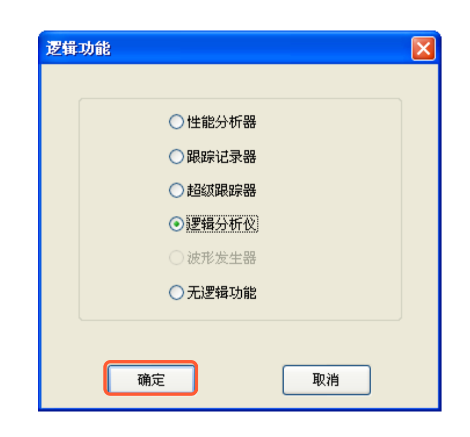


图 6.24　逻辑分析仪设置窗口

6.3.2 实际使用方法

用户程序编译通过，TKScope 仿真器正确设置完成之后，即可开始仿真程序并使用内置的 32 路逻辑分析仪分析总线上的数据。

1 点击【**Debug**】进入仿真状态，然后双击 zlgLogic.exe 打开逻辑分析仪软件。打开的逻辑分析仪软件如图 6.25 所示，通讯正常的状态，右下角显示“在线”，否则显示“离线”。

注意！ 一定要先进入仿真状态，再打开逻辑分析仪软件，这样才是与仿真状态结合使用。如果，在程序进入仿真状态之前打开逻辑分析仪软件，即作为独立的逻辑分析仪使用，此时，程序无法进入仿真状态。

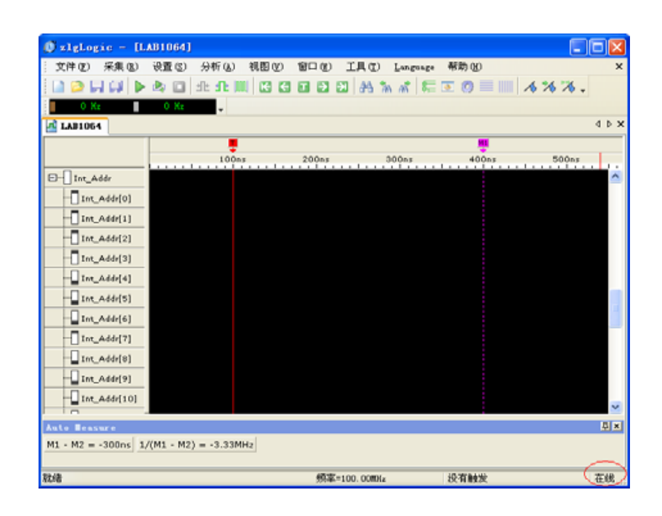


图 6.25　逻辑分析仪界面

2 点击【**Run**】程序全速运行起来，此时用户可以按照实际需要设置触发条件、采样频率等参数，然后满足触发条件时逻辑分析仪自动捕捉总线上的数据。



此时，逻辑分析仪的设置、控制等方法和独立的逻辑分析仪一样，在此不做详细讲解，用户可参考逻辑分析仪的用户使用手册。

3 例程是反复循环的操作过程，我们可以采用最简单的方法来捕获总线上的数据。程序全速运行起来时，点击逻辑分析仪主界面的 按钮，循环采集，直到点击 停止。图 6.26 就是逻辑分析仪捕捉到的总线数据图。

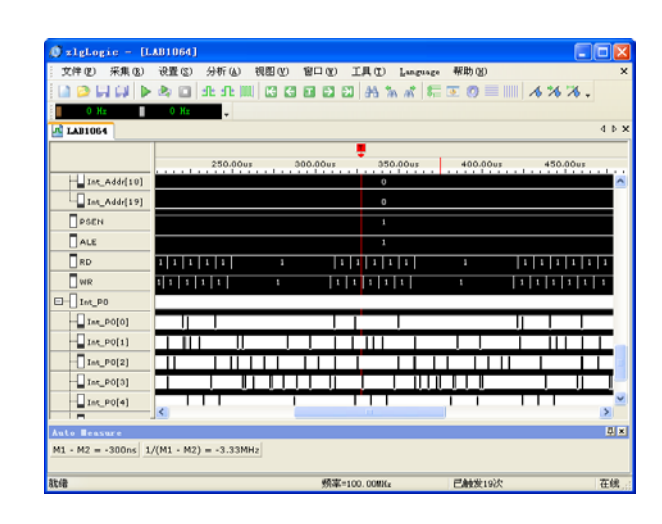
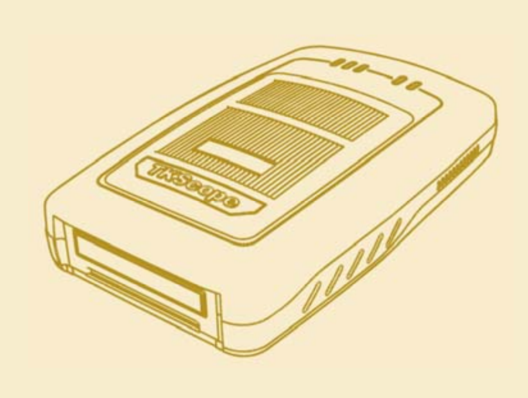


图 6.26　逻辑分析仪的显示结果

4 **结果分析**

例程是在 P89V51RD2FA 芯片中进行仿真的，其内部有 768 字节的 XDATA 空间，与外部 XDATA 空间前 768 字节地址相同。当 AUXR.1=0 时，访问内部 XDATA 空间，此时 WR、RD 无效；当 AUXR.1=1 时，访问外部 XDATA 空间，此时 WR、RD 有效。

例程是先对内部 XDATA 空间的前 6 个字节进行写读操作，然后对外部 XDATA 空间的前 6 个字节进行写读操作。读写信号在前半段是无效的，后半段是有效的。如图 6.26 所示，是逻辑分析的总线数据显示图。这里主要针对 WR、RD 信号线进行分析，可以很清楚的看到显示结果是完全正确的。同样，也可以查看其它总线的数据，来分析程序的执行状态和总线上的数据。



第 7 章

On the fly 运行中操作

7.1 观察存储空间

On the fly 运行中操作是 TKScope 仿真器独具的特色功能，目前在业界内只有 TKScope 仿真器能够实现在运行中操作的功能，是仿真技术的一项重大突破。

On the fly 运行中操作的具体含义是：

- 运行中观察芯片内部的全部寄存器，如 A、B、DPTR、SP、R0~R7 等。
- 运行中观察芯片内部的 data/idata 空间。
- 运行中观察 xdata 空间并可修改数值。
- 运行中观察 code 空间并可修改数值。
- 运行中观察程序消耗时间的变化。
- 运行中观察代码执行覆盖分析实时结果。
- 运行中观察性能分析实时结果。
- 运行中设置/取消断点。



注意！上述的操作是在进入仿真状态后，程序执行全速运行过程中进行的操作，而不是其它的仿真器进入仿真状态后，程序停止运行时进行的操作。

On the fly 运行中操作这种仿真技术，允许用户在程序运行过程中查看/修改全部资源，程序运行对于客户来说完全的透明化。程序运行每一步的执行结果，用户通过查看相应的存储空间即可一目了然；程序运行的轨迹，函数、变量的执行情况，用户通过观察性能分析器和代码覆盖分析、统计结果即可了如指掌。可以说 On the fly 技术揭去了程序运行的神秘面纱，让用户清清楚楚的看到了程序运行的真正面目。而且，用户在程序运行过程中可以按照自己的意愿任意的修改程序代码，任意的设置或取消断点，完全控制程序运行于股掌之间，让程序运行更加灵活，用户分析程序更加得心应手。

On the fly 运行中操作是 TKScope 仿真器独创的一项仿真领域的新技术，对于它的使用方法，用户可能还很生疏，下面将详细讲述使用方法，让用户感受到这种技术给程序分析带来的方便性、高效性、灵活性。

运行中观察寄存器

1 运行中观察寄存器

运行中可以动态观察芯片内部的全部寄存器，方法很简单。

常用的寄存器，如 A、B、DPTR、SP、R0~R7 等，在主界面左侧【**Register**】窗口可直接观看到动态变化值。其它的寄存器只需打开相应的寄存器观察窗口即可看到动态变化值。

如图 7.1 所示，在【**Peripherals**】菜单下，用户可以根据实际观察需要进行选择，弹出的观察窗口中有相关的寄存器。如用户需要观察与中断相关的寄存器，要选择【**Interrupt**】选项，弹出的观察窗口如图 7.2 所示，此时用户可以查看相关寄存器的动态变化值。在【**Peripherals**】菜单下，用户还可以选择 I/O 口、串口、定时器、SPI 总线等，观察与之相关的寄存器。

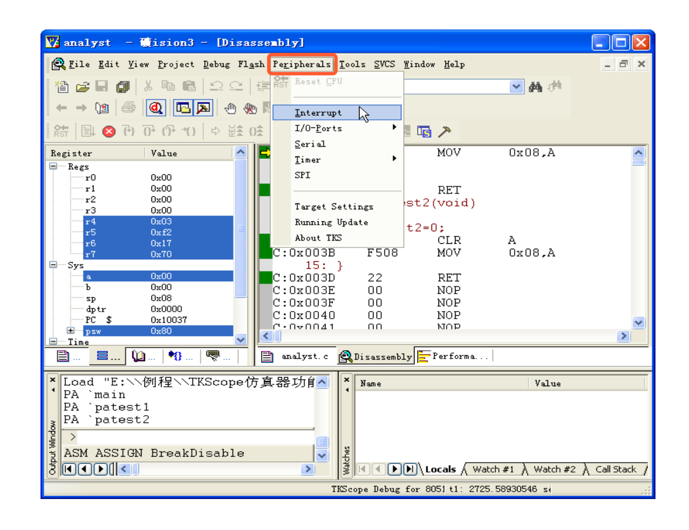


图 7.1　选择需要观察的外设选项

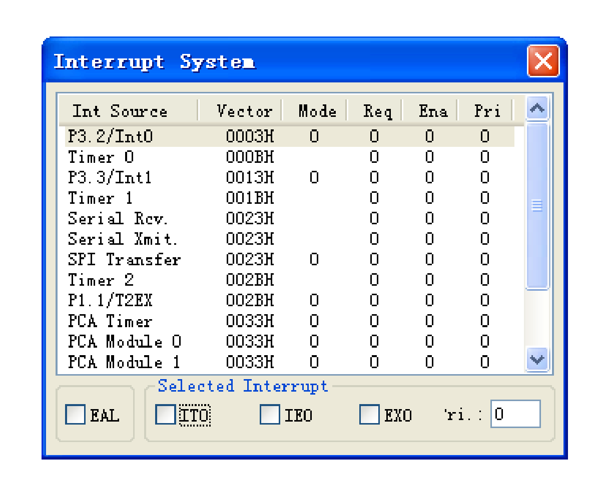


图 7.2　中断的寄存器观察窗口

2 运行中观察 data、idata、xdata、code 空间

选择【View】菜单下的【Memory Window】选项，打开存储器窗口，如图 7.3 所示。

在存储器窗口中输入需要观察的存储空间即可观察该空间的数据。如需要观察 Data 空间 12H 地址之后的数据，只需输入“d:0x12”或“d:12H”，如图 7.4 所示。

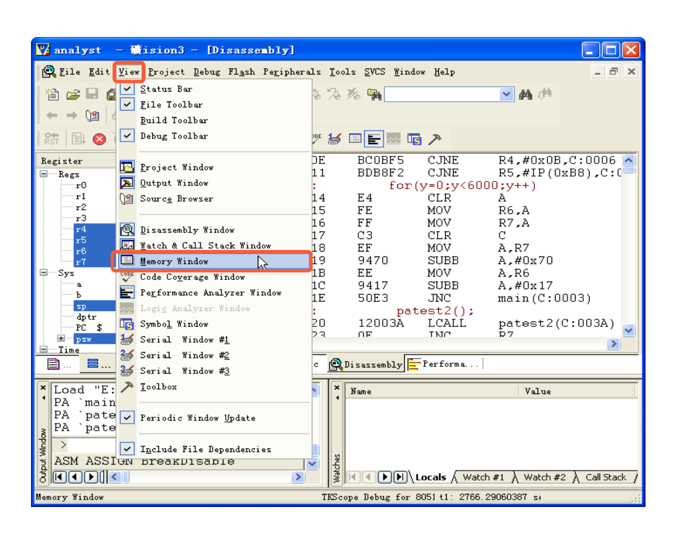


图 7.3　打开存储器窗口

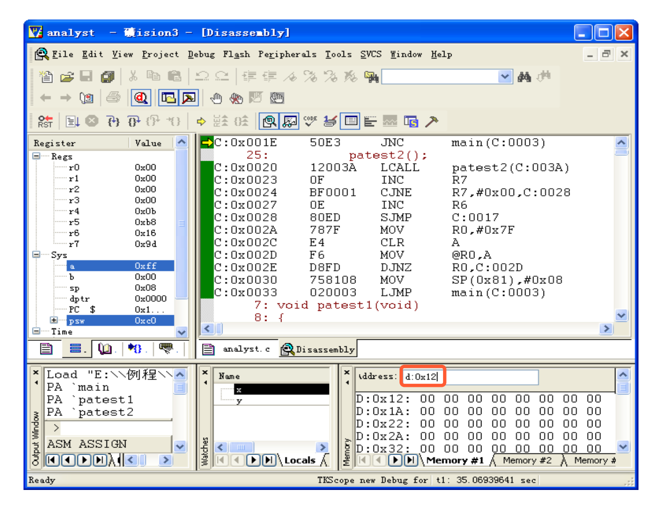


图 7.4　观察 Data 空间

观察存储空间的输入方法规则

Data 空间	d:0xXX 或 d:XXH
Idata 空间	i:0xXX 或 i:XXH
Xdata 空间	x:0xXXXX 或 x:XXXXH
Code 空间	c:0xXXXX 或 c:XXXXH



用户如果还需要观察其它存储空间的数据，可以选择右侧的【Memory＃2】窗口，输入相应的空间值即可。Keil IDE 软件最多支持 4 个存储窗口同时观察。

7.2 修改存储空间

程序运行过程中，用户可以修改 xdata、code 空间的值，修改是在存储器窗口中进行的。

1 方法

方法很简单，鼠标移到需要修改的数据位置，点击鼠标右键，在弹出的菜单中选择【**Modify Memory at**】选项，然后在弹出的对话框中输入要修改的值即可。

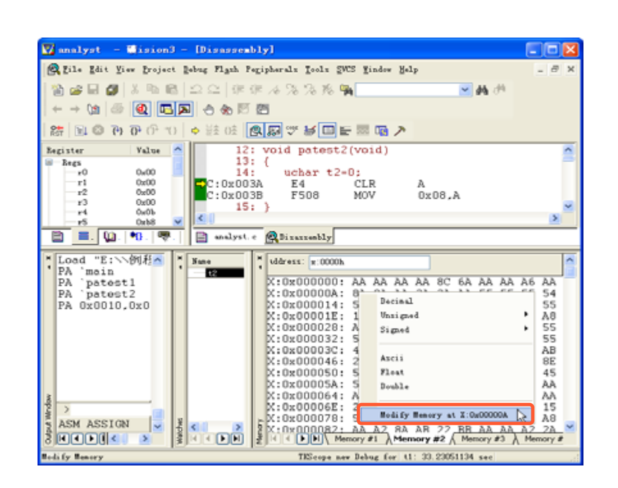


图 7.5　选择修改选项

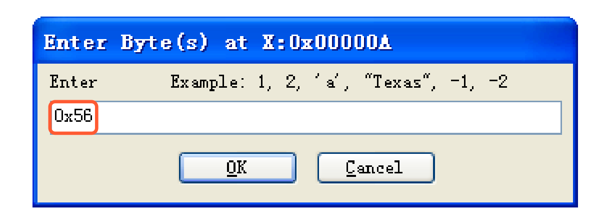


图 7.6　输入需要修改的数据

2 实例

例如，用户需要修改 xdata 空间 0x000A 地址的数据为 0x56。鼠标移到 xdata 区域 0x000A 地址处数据所在的位置，点击鼠标右键，在弹出的菜单中选择【**Modify Memory at X:0x00000A**】选项，如图 7.5 所示。此时系统会弹出对话框，在此对话框中输入 0x56，然后点击【**OK**】即可完成修改数据，如图 7.6 所示。数据修改完毕，在 xdata 区域 0x000A 地址处可以看到数据是 0x56，如图 7.7 所示。

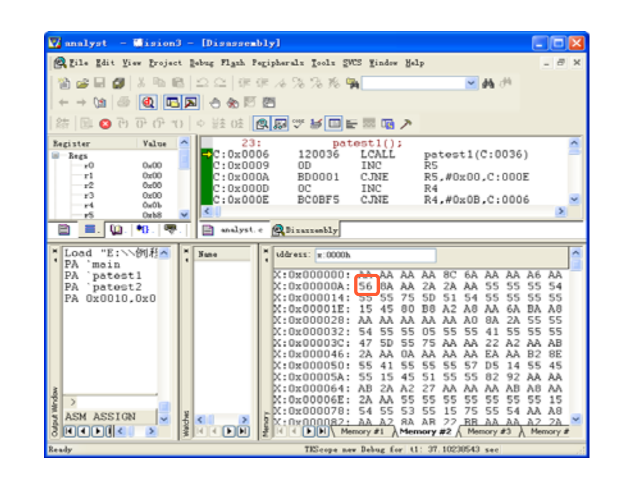


图 7.7　数据修改结果

7.3 观察消耗时间

程序运行过程中，用户可以观察程序消耗时间的动态变化值。如图 7.8 所示，在主界面的【**Register**】窗口中，可以看到时间的变化情况。

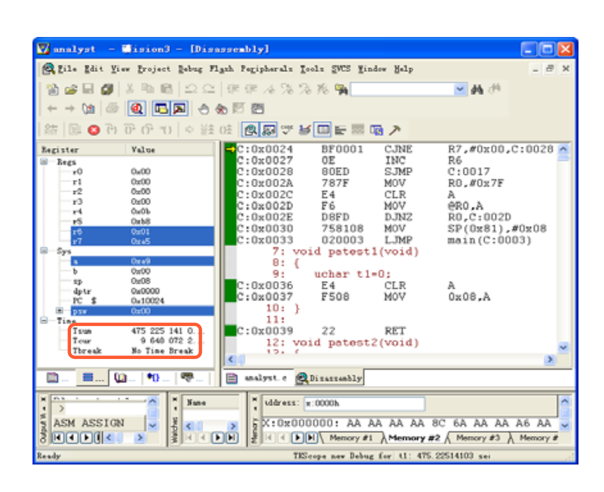


图 7.8　动态观察程序运行时间

1 【总运行时间 Tsum】

仿真器从上电或复位后，到当前状态经历的有效运行总时间。Tsum 是有效的运行时间的累积，程序处于仿真状态但停止运行时，时间不累积。

2 【当前运行时间 Tcur】

记录当前一次有效运行操作经历的时间。

例如，运行一个单步经历了 1us，则 Tcur 显示为 1us；再运行一个单步经历了 3us，则 Tcur 显示为 3us。与总运行时间的不断累积不同，Tcur 是个时间差，便于用户观察本次操作经历的时间。

7.4 观察代码分析

程序运行过程中，可以观察代码执行覆盖分析，系统把程序中的各个功能函数的执行情况，以百分比的形式显示出来，便于用户直观的观察到代码的执行覆盖情况。

选择【**View**】菜单下的【**Code Coverage Window**】选项，如图 7.9 所示，即可打开代码执行覆盖分析窗口，程序运行过程中通过此窗口观察代码执行覆盖情况。

针对这个功能模块，有篇专门的文档详细的讲解观察方法，用户请详见《第 4 章—代码分析》，在此仅作上述简单的观察方法介绍。

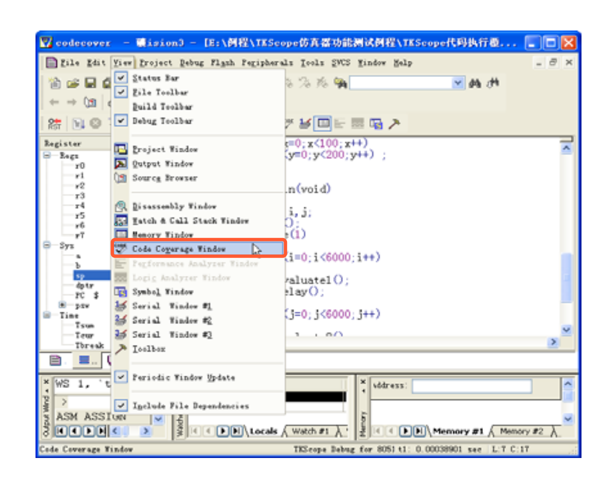


图 7.9　打开代码执行覆盖分析窗口

7.5 观察性能分析

程序运行过程中，可以观察函数、变量等的性能分析实时结果，系统以统计图形的方式显示出来，便于用户直观的观察到性能分析的结果。

选择【**View**】菜单下的【**Performance Analyzer Window**】选项，如图 7.10 所示，即可打开性能分析显示窗口，程序运行过程中通过此窗口观察性能分析的结果。

针对这个功能模块，有篇专门的文档详细的讲解定义方法和观察方法，用户请详见《**第 6 章—性能分析器的使用方法**》，在此仅作上述简单的观察方法介绍。

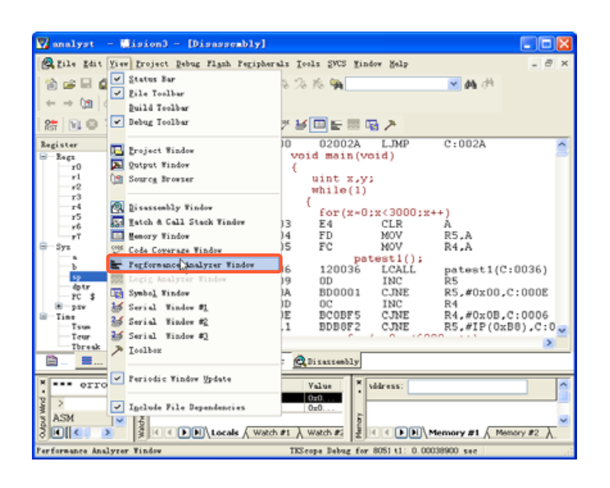


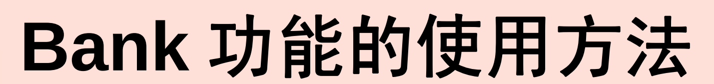
图 7.10　打开性能分析显示窗口

7.6 设置、取消断点

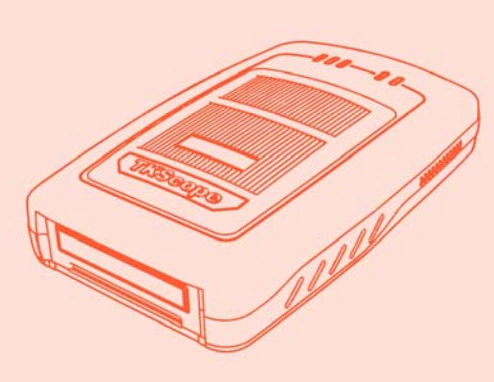
程序运行过程中，用户可以根据仿真需要进行设置、关闭、取消断点等操作。

关于断点的种类和操作方法，用户请详见《**第 5 章—断点操作技巧**》，文中详细的介绍了 TKScope 仿真器断点的种类、操作方法。在此不作详细描述，只强调 TKScope 仿真器断点的操作是可以在程序运行中进行的，这就是 On the fly 的特性表现。

第 8 章



Bank 功能的使用方法



8.1 Bank 功能原理介绍



8051 的代码寻址空间和数据寻址空间最大为 64KB。如果用户的代码和数据长度超过 64KB，就必须使用 Bank 技术来实现。在编译器的支持下，理论上 Bank 技术可以达到无穷大的代码空间和数据空间，而且用户完全不需要考虑不同 Bank 之间的切换问题。

本章将详细讲解 Bank 功能的原理以及在 Keil 环境下使用 TKScope 仿真的方法。

8.1.1 功能原理

Bank 功能的实现原理非常简单，就是把 **64KB** 空间分成公用空间和分组空间。

公用空间用于实现分组切换和一些特殊用途，如中断、快速操作函数等。在任何条件下，公用空间总是占据着 64KB 寻址空间的开始地址，尺寸一般选择为 32KB。

分组空间用于存放代码或固定数据，一般占据着公用空间后面的位置，尺寸选择为 32KB 比较理想。与公用空间不同，在寻址分组空间的地址时，可以选择多个分组，这样就扩展了可寻址范围。

合适的编译器可以调度不同 Bank 的切换，但是用户必须自己决定不同 Bank 之间如何切换。**Bank 切换**可以由多种方法来实现，例如重造一个位于 Xdata 的 I/O 地址来实现，或使用 8051 的 I/O 口来实现。



实现 Bank 功能必须注意下面的问题！

- 选择合适的编译器以及正确的设置。建议使用 Keil 公司的 uVision3 环境。
- 选择支持 Bank 功能的硬件仿真器。TKScope 仿真器的 K9 型号支持最大 2×512KB 分组空间。

8.1.2 环境设置

在 Keil uVision3 环境下实现 Bank 功能，用户需要在应用前对编译环境的有关参数进行设置。本小节将着重指出需要设置的参数，以及各项参数的意义。



注意！ 用户必须按照实际的应用情况设置参数，否则 Bank 功能可能无法正确使用。

1 选择【**Project**】菜单下【**Options for Target** 】选项或点击快捷图标 ，进入如图 8.1 所示的界面。

2 点击图 8.1 中的【**Target**】选项，进入如图 8.2 所示的界面。

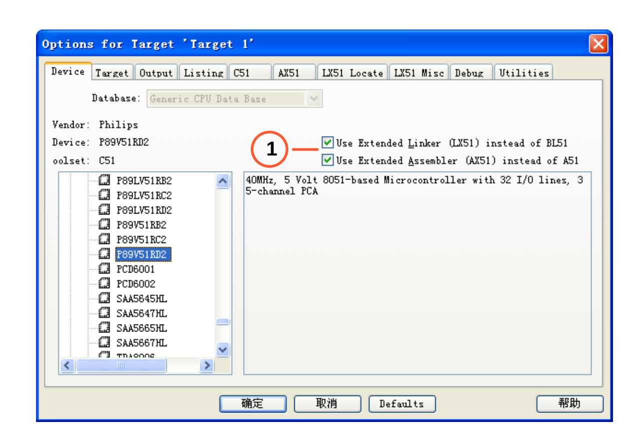


图 8.1　Bank 功能参数设置界面 1

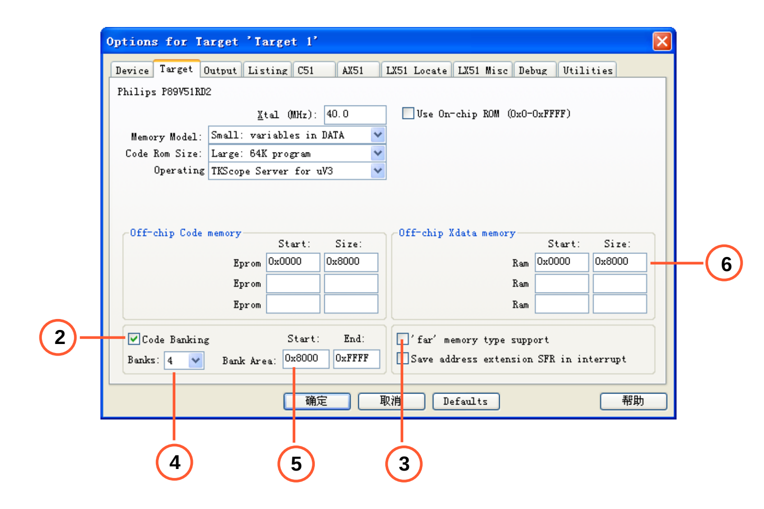


图 8.2　Bank 功能参数设置界面 1

①　选择数据分组模式的编译

采用数据分组操作时，此选项必须选择。因为数据分组要用到 far 变量，必须使用 **Ax51** 和 **Lx51** 选项。

far 变量可能是 const 和数据变量。例如：

unsigned char far large_array【0x8000】;　　　　　　//**xdata** 变量

const unsigned char far large_carry【0x8000】;　　　　//**code** 变量

②　选择代码分组模式

使用代码分组时，需要选中。

③　选择 xdata 分组模式

使用数据分组时，需要选中。

⑥　选择 xdata 公共区域

选择 xdata 的公共区域。根据分组原则，公共区域一般放在地址的顶部，范围为全部地址的一半。

xdata 在分组时同 code 分组一样，也可以有一个不分组的公共区域。该区域在任何分组中都可以读取。分组地址的变化不会影响操作公共区域。

如果不填写无公共区域，64KB 地址范围内在仿真器硬件上都是任何分组的。

④　选择代码分组个数

使用代码分组时，需要选择分组的个数，有 2/4/8/16 等选择。选择分组个数时，要兼顾硬件仿真器的情况，如 **TKScope** 仿真器**最大**有 **8 个分组**，在这里尽量选择 8。

⑤　选择代码分组区域

选择分组区域的起始地址和结束地址，这是一个非常重要的参数，尤其在硬件仿真时一定要正确填写。

Bank Area 选项中的 Start 起始地址以上的地址范围，称作公共区域，用于分组切换。该区域的代码不分组，在任何分组地址下代码都是一致的，用户的硬件必须满足这个条件，可以使用 MCU 内部的存储区域，例如 P89C58 内部 32KB 的代码空间。

分组区域可以有很多组，数量由④中的代码分组数决定。

3

点击图 8.1 中的【LX51 Locate】选项，进入如图 8.3 所示的界面。

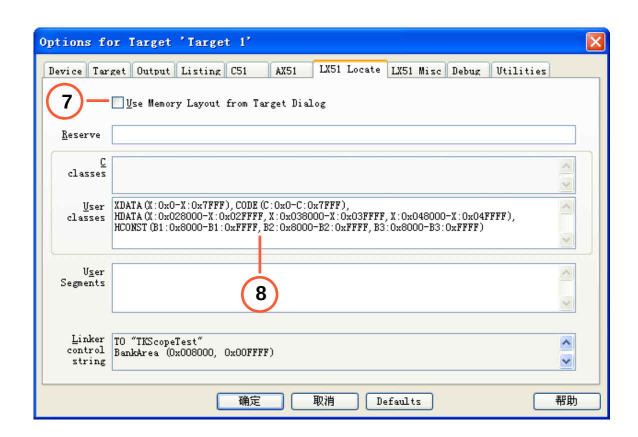


图 8.3　Bank 功能参数设置界面 3

⑦ **选择使用连接器参数**

如果用户选择数据分组，此项一定不要选中。

⑧ **输入连接器参数**

输入 far 参数需要的分组，例如：

Xdata（X:0x0-X:0x7FFF）

Code（C:0x0-C:0x7FFF）

HDATA(X:0x028000-X:0x02FFFF, X:0x038000-X:0x03FFFF, X:0x048000-X:0x04FFFF)

HCONST(B1:0x8000-B1:0xFFFF, B2:0x8000-B2:0xFFFF, B3:0x8000-B3:0xFFFF)

4

选中程序文件，点击鼠标右键，在弹出的菜单中选择【Options for File】选项，如图 8.4 所示。

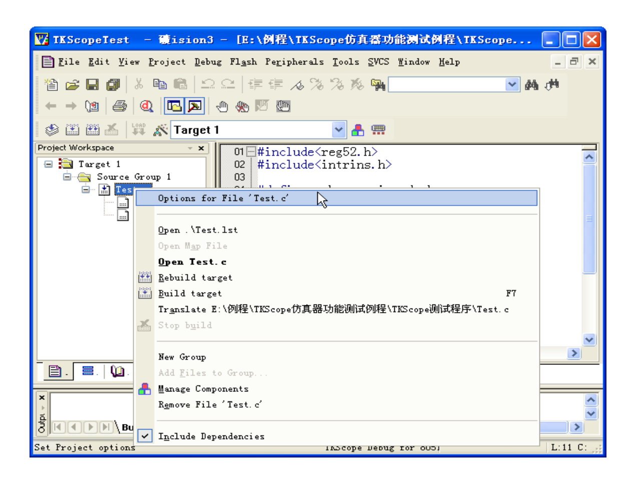


图 8.4　打开文件配置选项



注意！HCONST 定义了 code 的分组，但不是程序代码的分组。

⑨ **为文件模块选择组号**

用户根据实际要求为选中的文件选择组号。

5

打开的主文件配置选项如图 8.5 所示。

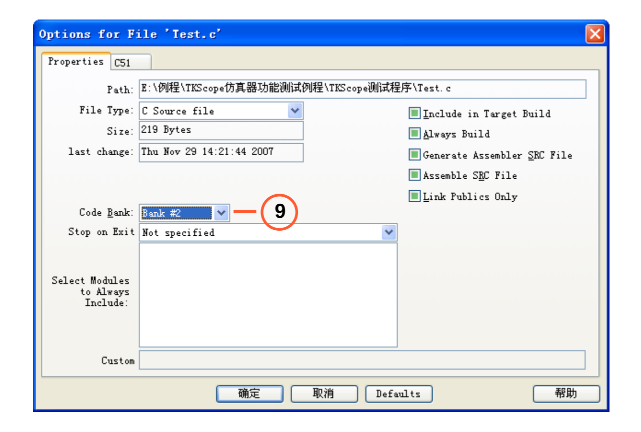


图 8.5　主文件配置选项

8.1.3 配置文件

Keil uVision3 编译器需要知道用户的**分组**情况，尤其需要知道用户分组**切换**的方式，要求用户以**文件**的方式完成。



在 uV3 中有一个模板文件，名称是 L51_BANK.A51，位于 C51/LIB 目录下。使用 Bank 功能时，用户必须将该文件加入工程中，并根据实际的硬件情况进行修改。

L51_BANK.A51 文件中决定 **Bank 模式**的参数如下：

```
?B_NBANKS        EQU    4        ;定义最大分组个数，必须与实际硬件相对应
?B_MODE          EQU    1        ;0 表示通过 8051 的 I/O 口进行分组切换
                                 ;1 表示使用 XDATA 地址进行切换
?B_VAR_BANKING   EQU    0        ;是否使用该文件进行数据分组
                                 ;0 不使用该文件
                                 ;1 使用该文件
?B_RST_BANK      EQU    0xFF     ;复位后的 Bank 组号
```

L51_BANK.A51 文件中决定 **Bank 分组切换**的参数如下：

```
Bank 应用模式 0 时的参数
IF   ?B_MODE = 0
P1               DATA   90H      ;用户可以改为其它 I/O 口地址，例如 P3
?B_PORT          EQU    P1
?B_FIRSTBIT      EQU    0        ;用于分组切换的第一个位
ENDIF

Bank 应用模式 1 时的参数
IF   ?B_MODE = 1
?B_XDATAPORT     EQU    0FFFFH   ;用于分组切换的 XDATA 地址
?B_FIRSTBIT      EQU    0        ;用于分组切换的第一个位
ENDIF
```

8.2 Bank 功能仿真方法

TKScope 仿真器的 **K9** 型号**独具** Bank 分组仿真功能！

- 突破 8051 的 64KB 代码数据限制，可仿真最大 8 分组的 64KB 空间。
- 支持代码分组调试，最大 8×64KB。
- 支持数据分组调试，最大 8×64KB。
- 支持 4 路仿真器外部 Bank 调试信号输入。
- 支持无限制数量 MCU 内部 Bank 控制信号。

用户在软件上设置完成之后，就可以使用 TKScope **K9** 仿真器进行实际的硬件仿真了。但是，**Bank 功能模式**的仿真与非分组模式的仿真不同，采用 Bank 模式仿真，用户**必须**自行输入硬件**分组地址**。



POD-8051HS-P84 仿真头支持 Bank 模式的仿真，该 POD 头上有 4 个外部 Bank 地址输入，用户需要把目标板上产生的 Bank 切换信号由这里输入到仿真器内部。

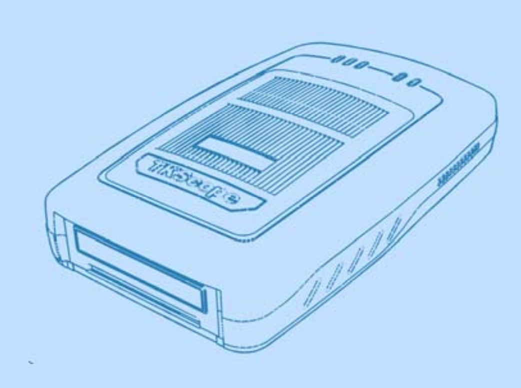


注意！理论上，4 个外部地址输入端最大支持 16 分组，但是，实际上，K9 的硬件最大支持 8 分组。用户输入 Bank 分组切换信号时，4 个输入端一定要全部有信号输入，不可悬空，否则会造成仿真错误或失败！

TKScope 仿真器的 **K9** 支持无限制数量 MCU 内部 Bank 控制信号。仿真芯片内部 Bank 分组时，无需在外部输入切换信号，只需在内部分组选项里根据实际仿真需要设置即可，具体请详见《**第 3 章—内部分组**》。

第 9 章





9.1 常见问题

初次使用 TKScope 仿真器的用户，经常会遇到一些使用上的问题，下面列举出一些用户可能会经常提到的问题，供初学者参考。

问题 为什么我的仿真器无法联机，不能正常使用。

现象 初次使用 TKScope 仿真器，安装完毕驱动之后，仿真器不能联机使用。

原因 绝大多数情况是没有正确的安装驱动，系统识别不到新的 USB 硬件设备。
用户需参考《第 2 章－驱动安装方法》，正确安装驱动，让系统识别到 USB 硬件设备。

问题 为什么我的仿真器无法进入监控状态。

现象 TKScope 仿真器正常联机使用，但是点击 Debug 调试按钮，无法进入监控状态。

原因 TKScope 仿真器没有正确的设置，没有正确指定当前仿真器的具体型号。建议用户根据实际仿真情况选择正确的仿真芯片种类之后，利用系统提供的搜索功能得到仿真器的具体型号，然后保存配置信息。
用户可参考《第 3 章－硬件选择》。

问题 为什么我在仿真时总线总是出现异常波动。

现象 单步运行时，程序没有总线操作，或者是操作片内 XRAM，进入监控后，总线总是出现波动的现象。由于 P0/P2 口用于 I/O 功能，出现波动破坏了我的电路状态。

原因 在 IDE 环境里打开了 Memory 窗口，用于显示 XDATA 的数据，由于监控程序需要读取 XDATA 数据进行显示，因此造成总线出现波动，关闭 XDATA 存储区显示即可解决。

问题 为什么我在仿真时，第一次运行后程序总是停在一个固定位置。

现象 仿真程序过程中，开始运行后程序总是停在某一个位置，需要再次启动运行后才能持续运行。检查该位置并没有设置断点。

原因 很有可能是设置了时间断点，首次运行后超过时间断点数值，程序中止运行进入监控状态，再次运行后该时间断点不再满足条件，所以程序可持续运行。可以取消时间断点，把时间断点的数值改为 0 即可。

问题 为什么我的仿真器突然失效，仿真不正常。

现象 仿真器之前工作一切正常，插拔之后突然仿真混乱，程序运行起来 PC 指针跑到程序的无效代码区。

原因 看似很奇怪很复杂的问题，其实很简单的原因，就是仿真头没有接触好。与仿真器连接的排线以及仿真头各个组件出现接触不良、短路、断路等情况都会导致仿真器产生故障现象。

问题 为什么我在使用 Bank 分组时，运行代码总是出错。

现象 运行公共区域和 Bank0 代码正确，但是运行 Bank1 以上区域的代码总是出错。

原因 主要是存储器影像设置不对。TKScope 仿真器的缺省设置是 64K，满足一般情况下 8051 的仿真。在 Bank 分组模式下，代码区域为多个 64K，因此需要设置足够多的存储器区域到仿真器内部，或者干脆选择 All HardWare ROM Internal。

问题 为什么我在仿真 C 语言程序时，总是进入反汇编窗口。

现象 仿真 C 语言程序时，全速或单步运行一条 C 语言程序行，总是进入反汇编窗口。

原因 TKScope 仿真器支持运行中刷新仿真资源，包括 PC 指针。当全速或单步运行期间，仿真器要读取运行中 PC 指针数据，并在仿真界面上显示出来。由于 PC 指针数据并不能完全对应 C 程序行，C 语言窗口无法显示，只能转到或打开反汇编窗口进行显示。
解决的方法是：打开【Perpherals】菜单下的【Running Update】窗口，取消 pc arrow 选项。

问题 为什么我选用 No Bus 总线模式时，ALE 无脉冲输出。

现象 选用 No Bus 总线模式仿真，ALE 无脉冲输出，总是停留在高电平。

原因 在 No Bus 总线模式下，P0/P2 口总是表现为 I/O 口模式，即使遇到 MOVX 指令，ALE 信号也没有输出变化，停留在高电平上。这种表现能获得较好的电磁特性。
如果用户需要使用 ALE 信号，请选择 Only Xdata Bus。

9.2 联系我们

感谢您选用 TKScope 仿真器，我们将竭诚为您提供技术服务，帮助您解决在使用仿真器开发过程中遇到的问题。为了尽快解决问题，不影响您的开发进度，我们建议您通过下面的方式联系我们，获得技术支持。

1 电话（传真）

使用电话沟通交流是最快的解决问题的途径。请您在拨通电话之前，整理好您的问题，将仿真器放置在计算机和电话机旁边并处于联机状态，我们的技术支持工程师会根据您的操作来分析您提出的问题。

技术支持电话：020-22644360 22644361。

2 电子邮件

使用电子邮件您可以详细的说明描述您遇到的问题，也可以提供测试程序或工程文件等，我们更有针对性的帮您分析解决问题。

技术支持 Email：TKS@zlgmcu.com。

3 BBS

欢迎访问周立功单片机论坛 http://www.zlgmcu.com.cn/index.asp，我们为仿真器开设了专门的 BBS，并安排工程师轮流值日制度，对于用户提出的全部问题，可以得到我们详尽的技术解答，同时也可以得到其他有经验的网友朋友的回答。

4 维修服务

如果您在使用过程中遇到仿真器损坏的现象，请寄给广州致远电子有限公司维修部，由生产厂商直接快捷的为您服务，详细的联系方式如下：

公司：广州致远电子有限公司

地址：广州市天河区车陂路黄洲工业区 3 栋 2 楼

电话：020-22644245

注意！由于仿真器属于特殊产品，需要用户在开发环境里正确的设置才能正常的使用，所以多数情况下仿真器出现问题可能是设置不正确引起的，而不是仿真器损坏，用户不需要返修而解决问题。



友情提醒！如果您认为仿真器出现问题，我们建议您在决定返修前，与我们的技术支持工程师进行联系，确认仿真器是否真正的出现硬件上的故障需要维修。另外，您需要将仿真器的故障现象和您的详细联系方式写成文字资料，与仿真器一同返回，这样我们的维修人员能根据您提供的资料快速的进行处理，节省您的维修时间。

9.3 结束语

TKScope 仿真器是广州致远电子有限公司在 2008 年推出的一款全新概念的高级专业仿真开发平台，无论是性能还是外观，都经过工程师的精心设计，将带给用户一个全新的开发理念和感受。

TKScope 仿真器使用当今超大规模集成电路的最新技术以及规范的硬件模块化设计，全面提升 TKScope 仿真器在嵌入式系统开发中的表现能力。

TKScope 仿真器突破以往仿真器所不能，采用多种先进的仿真技术，与多种主流 IDE 环境无缝嵌接，集成众多强悍有力的功能，可仿真芯片种类之多之广，让其不愧为仿真器界的明星产品。

再次感谢您选用 TKScope 仿真器，希望能够带给您帮助和惊喜，给您不一样的开发感觉，让您如虎添翼，项目早日成功！

如果您遇到关于嵌入式系统的任何问题，请联系我们的技术支持工程师或访问我们的主页 http://www.zlgmcu.com。从产品的设计思想、元器件的选型、开发工具的使用、可靠性的设计等各个方面，我们的技术支持工程师都能为您提供全套的解决方案或良好的建议。

附录 TKScope 仿真器支持芯片列表

TKScope 仿真器支持众多厂商和海量芯片的仿真，而且随着新型号器件的出现不断升级，满足客户的全方面需求。目前，TKScope 仿真器已经或近期陆续支持的芯片明细列表如下，厂商按照字母顺序排列，排名不分先后。

- **Acer Labs**

8051 内核 M6759
其它型号芯片陆续支持

- **ACTEL**

Cortex-M1 内核

- **Aeroflex UTMC**

8051 内核 UT69RH051
其它型号芯片陆续支持

- **AMD**

Flash 器件 AM29F160DB，AM29F160DT，AM29F320DB，AM29F320DT，AM29F800BB，AM29F800BT，AM29LV128，AM29LV800BB，AM29LV800BT，AM29LV800DB
其它型号 Flash 器件陆续支持

- **Analog Device**

8051 内核 ADuC812，ADuC816，ADuC824，ADuC831，ADuC832，ADuC834，ADuC836，ADuC841，ADuC842，ADuC843，ADuC845，ADuC847，ADuC848
其它型号芯片陆续支持

ARM 内核 ADuC7019-62，ADuC7020-62，ADuC7021-62，ADuC7021-32，ADuC7022-62，ADuC7022-32，ADuC7024-62，ADuC7025-62，ADuC7025-32，ADuC7026-62，ADuC7027-62，ADuC7028-62
其它型号芯片陆续支持

- **ARM**

ARM 内核 ARM720T，ARM7EJ-S，ARM7TDMI，ARM7TDMI-S，ARM9TDMI，ARM920T，ARM922T，ARM926EJ-S，ARM946E-S，ARM966E-S，ARM968E-S，ARM1136，ARM1156，ARM1176，Cortex-M0，Cortex-M1，Cortex-M3
其它内核种类陆续支持

- **ATMEL**

8051 内核 AT80C31X2/TS80C31X2，AT80C32X2/TS80C32X2，AT80C5112，AT80C51RA2/TS80C51RA2，AT80C51RB2/TS80C51RB2，AT80C51RC2/TS80C51RC2，AT80C51RD2/TS80C51RD2，AT80C51SND1C，AT80C51SND2C，AT80C51X2/TS80C51X2，AT80C52X2/TS80C52X2，AT80C54X2/TS80C54X2，AT80C58X2/TS80C58X2，AT80SUND2CMP3B，AT83C5111，AT83C5112，AT83C5121/T83C5121，AT83C5122，AT83C5123，AT83C5127，AT83C51RA2/TS83C51RA2，AT83C51RB2/TS83C51RB2，AT83C51RC2/TS83C51RC2，AT83C51RD2/TS83C51RD2，AT83C51SND1C，AT83C51SND2C，AT83EC5122，AT83EC5123，AT83SUND2CMP3B，AT85C5121/T85C5121，AT85C5122，AT85C51SUND3B1，AT85C51SUND3B2，AT85C51SUND3B3，AT87C5111，AT87C5112，AT87C51RA2/TS87C51RA2，AT87C51RB2/TS87C51RB2，AT87C51RC2/TS87C51RC2，AT87C51RD2/TS87C51RD2，AT87C51X2/TS87C51X2，AT87C52X2/TS87C52X2，AT87C54X2/TS87C54X2，AT87C58X2/TS87C58X2，AT87F51，AT87F52，AT89C1051，AT89C2051，AT89C4051，AT89C51，AT89C5115/T89C5115，AT89C5121/T89C5121，

AT89C5122，AT89C5122DS，AT89C5130A-L，AT89C5130A-M，AT89C5131A-L，AT89C5131A-M，AT89C5132，AT89C51AC2/T89C51AC2，AT89C51AC3，AT89C51IC2，AT89C51ID2，AT89C51CC01/T89C51CC01，AT89C51CC02/T89C51CC02，AT89C51CC03，AT89C51RA2，AT89C51RB2，AT89C51RC2，AT89C51RC，AT89C51RD2，AT89C51RE2，AT89C51ED2， AT89C51SUND1C，AT89C51SUND2C，AT89C52，AT89C55WD，AT89SND2CMP3B，AT89LS51，AT89LS52，AT89LS53，AT89LS8252，AT89LV51，AT89LV52，AT89LV55，AT89S51，AT89S52，AT89S53，AT89S8252，AT89S8253，TS80C51U2，T83C5101，T83C5102，T83C51U2，T87C5101，T87C5102，T87C51U2，T89C51RB2，T89C51RC2，T89C51RD2

其它型号芯片陆续支持

ARM 内核 AT91SAM7A1，AT91SAM7A2，AT91SAM7A3，AT91SAM7SE32，AT91SAM7SE256，AT91SAM7SE512，AT91SAM7S16，AT91SAM7S161， AT91SAM7S32，AT91SAM7S321， AT91SAM7S64，AT91SAM7S128，AT91SAM7S256， AT91SAM7S512， AT91SAM7X128， AT91SAM7X256， AT91FR40162S， AT91FR40162SB，AT91M40800，AT91M40807，AT91R40008，AT91R40807

其它型号芯片陆续支持

Cortex-M3 内核

AT91SAM3U4E，AT91SAM3U2E，AT91SAM3U1E，AT91SAM3U4C，AT91SAM3U2C，AT91SAM3U1C

其它型号芯片陆续支持

AVR 内核 ATmega16，ATmega16A，ATmega162，ATmega164P，ATmega164PA，ATmega165，ATmega165P，ATmega165PA，ATmega169，ATmega169P，ATmega169PA，ATmega32，ATmega32A，ATmega323，ATmega324P，ATmega324PA，ATmega325，ATmega325P，ATmega3250，ATmega3250P，ATmega329，ATmega329P，ATmega329PA，ATmega3290，ATmega3290P，ATmega64，ATmega64A，ATmega640，ATmega644，ATmega644P，ATmega644PA，ATmega645，ATmega6450，ATmega649，ATmega6490，ATmega128，ATmega128A，ATmega1280，ATmega1281，ATmega1284P，ATmega2560，ATmega2561，AT90CAN32，AT90CAN64，AT90CAN128，AT90USB646，AT90USB647，AT90USB1286，AT90USB1287，ATmega48，ATmega48A，ATmega88，ATmega88A，ATmega168，ATmega168A，ATtiny13，ATtiny13A，ATtiny2313，ATtiny2313A，ATtiny24，ATtiny24A，ATtiny25，ATtiny261，ATtiny261A，ATtiny4313，ATtiny44，ATtiny44A，ATtiny45，ATtiny461，ATtiny461A，ATtiny84，ATtiny84A，ATtiny85，ATtiny861，ATtiny861A，ATtiny87，ATtiny167，AT90PWM2，AT90PWM2B，AT90PWM216，AT90PWM3，AT90PWM3B，AT90PWM316，AT90USB82，AT90USB162，ATxmega64A3，ATxmega128A1，ATxmega128A3，ATxmega192A3，ATxmega256A3，ATxmega256A3B，ATxmega16A4，ATxmega16D4，ATxmega32A4，ATxmega32D4，ATxmega64A1，ATxmega64D3，ATxmega128D3，ATxmega192D3，ATxmega256D3

其它型号芯片陆续支持

Flash 器件 AT29C1024，AT29LV1024，AT49BV162A，AT49BV162AT，AT49BV320，AT49BV320T，AT49BV321，AT49BV321T，AT49BV1604A，AT49BV1614A，AT49BV6416，AT49LV320，AT49LV320T，AT49LV321，AT49LV321T，AT49LV1614A

其它型号 Flash 器件陆续支持

- **Cirrus Logic**

ARM 内核 全系列 ARM 内核，即将支持

- **CML Microcircuts**

8051 内核 CMX850

其它型号芯片陆续支持

- **Cybernetic Micro**

8051 内核 P-51

其它型号芯片陆续支持

- **Dallas**

8051 内核 DS80C310，DS80C320，DS80C323，DS80C390，DS80C400，DS80C410，DS80C411，DS83C520，DS83C530，DS87C520， DS87C530，DS87C550，DS89C420，DS89C430，DS89C440，DS89C450

其它型号芯片陆续支持

- **Faraday**

ARM 内核 FA526，FA626，FIA320，FIE3360

其它型号芯片陆续支持

- **Freescale**

ARM 内核 MAC7101，MAC7106，MAC7111，MAC7112，MAC7116，MAC7121，MAC7122，MAC7126，MAC7131，MAC7136，MAC7141，MAC7142

其它型号芯片陆续支持

- **Honeywell**

8051 内核 HT83C51

其它型号芯片陆续支持

- **Hynix**

ARM 内核 HMS30C7202

其它型号芯片陆续支持

GMS90xx GMS90C31，GMS90C32，GMS90C320，GMS90C51，GMS90C52，GMS90C54，GMS90C56，GMS90C58，GMS90L31，GMS90L32，GMS90L320，GMS90L51，GMS90L52，GMS90L54，GMS90L56，GMS90L58

其它型号器件陆续支持

GMS97xx GMS97C1051，GMS97C2051，GMS97C51，GMS97C51H，GMS97C52，GMS97C52H，GMS97C54，GMS97C54H，GMS97C56，GMS97C56H，GMS97C58，GMS97C58H，GMS97L1051，GMS97L2051，GMS97L51，GMS97L52，GMS97L54，GMS97L56，GMS97L58，GMS99C58

其它型号器件陆续支持

- **Infineon**

8051 内核 C501G-1E，C501G-1R，C501G-L，C504-2E，C504-2R，C504-L，C505-2R，C505-L，C505A-2R，C505A-4R，C505A-4E，C505A-L，C505C-2R，C505C-L，C505CA-2R，C505CA-4R，C505CA-4E，C505CA-L，C505L-4E，C508-4E，C508-4R，C509-L，C511-R，C511A-R，C513-R，C513A-2R，C513A-H，C513A-R，C515-RM，C515-RN，C515-LM，C515-LN，C515A-4R，C515A-L，C515B-2R，C515C-8E，C515C-8R，C515C-L，C517A-4R，C517A-L，C540U-E，C541U-2E，SAB 80C515，SAB 80C515A，SAB 83C515A-5，SAB 80C517，SAB 80C517A，SAB 83C517A-5，SAB 80C535，SAB 80C537

其它型号芯片陆续支持

XC886 内核

XC864，XC866-1，XC866-2，XC866-4，XC878-13，XC878-16，XC886-6，XC886-8，XC888-6，XC888-8

其它型号芯片陆续支持

C166 内核 **XC166 系列**

XC161CJ_16F，XC161CS_32F，XC164GM_4F，XC164GM_8F，XC164GM_16F，XC164KM_4F，XC164KM_8F，XC164KM_16F，XC164TM_4F，XC164TM_8F，XC164TM_16F，XC164CM_4F，XC164CM_8F，XC164CM_16F，

XC164SM_4F，XC164SM_8F，XC164SM_16F，XC164LM_4F，XC164LM_8F，XC164LM_16F，XC164CS_8F，XC164CS_16F，XC164CS_32F，XC164D_8F，XC164D_16F，XC164D_32F，XC164N_8F，XC164N_16F，XC164N_32F，XC164S_8F，XC164S_16F，XC164S_32F，XC167CI_16F，XC167CI_32F

其它型号芯片陆续支持

XE166 系列

XE164F_24F，XE164F_48F，XE164F_72F，XE164F_96F，XE164G_24F，XE164G_48F，XE164G_72F，XE164G_96F，XE164H_24F，XE164H_48F，XE164H_72F，XE164H_96F，XE164K_24F，XE164K_48F，XE164K_72F，XE164K_96F，XE167F_48F，XE167F_72F，XE167F_96F，XE167G_48F，XE167G_72F，XE167G_96F，XE167H_48F，XE167H_72F，XE167H_96F，XE167K_48F，XE167K_72F，XE167K_96F，XE162FM_24F，XE162FM_48F，XE162FM_72F，XE162HM_24F，XE162HM_48F，XE162HM_72F，XE164FM_24F，XE164FM_48F，XE164FM_72F，XE164GM_24F，XE164GM_48F，XE164GM_72F，XE164HM_24F，XE164HM_48F，XE164HM_72F，XE164KM_24F，XE164KM_48F，XE164KM_72F，XE167FM_48F，XE167FM_72F，XE167GM_48F，XE167GM_72F，XE167HM_48F，XE167HM_72F，XE167KM_48F，XE167KM_72F，XE162FN_16F，XE162FN_24F，XE162FN_40F，XE162HN_16F，XE162HN_24F，XE162HN_40F，XE164FN_16F，XE164FN_24F，XE164FN_40F，XE164GN_16F，XE164GN_24F，XE164GN_40F，XE164HN_16F，XE164HN_24F，XE164HN_40F，XE164KN_16F，XE164KN_24F，XE164KN_40F

其它型号芯片陆续支持

XC2000 系列

XC2238M_56F，XC2239M_72F，XC2264_56F，XC2264_72F，XC2264_96F，XC2267_56F，XC2267_72F，XC2267_96F，XC2263M_56F，XC2263M_72F，XC2263M_104F，XC2264M_56F，XC2264M_72F，XC2264M_104F，XC2265M_56F，XC2265M_72F，XC2265M_104F，XC2267M_56F，XC2267M_72F，XC2267M_104F，XC2268M_56F，XC2268M_72F，XC2268M_104F，XC2285_56F，XC2285_72F，XC2285_96F，XC2286_56F，XC2286_72F，XC2286_96F，XC2287_56F，XC2287_72F，XC2287_96F，XC2285M_56F，XC2285M_72F，XC2285M104F，XC2286M_56F，XC2286M_72F，XC2286M_104F，XC2287M_56F，XC2287M_72F，XC2287M_104F，XC2336A_56F，XC2336A_72F，XC2365_48F，XC2365_56F，XC2365_72F，XC2361A_56F，XC2361A_72F，XC2363A_56F，XC2363A_72F，XC2364A_56F，XC2364A_72F，XC2364A_104F，XC2365A_56F，XC2365A_72F，XC2365A_104F，XC2387_72F，XC2385A_56F，XC2385A_72F，XC2385A_104F，XC2387A_56F，XC2387A_72F，XC2387A_104F，XC2735_40F，XC2735_72F，XC2765_72F，XC2765_104F，XC2766_96F，XC2785_72F，XC2785_104F，XC2786_96F

其它型号芯片陆续支持

- **ISSI**

8051 内核 IS80C31，IS80C32，IS80C51，IS80C52，IS80LV31，IS80LV32，IS80LV51，IS80LV52，IS89C51，IS89C52

其它型号芯片陆续支持

- **Luminary Micro**

Cortex-M3 内核

LM3S101，LM3S102 ，LM3S300，LM3S301，LM3S308，LM3S310，LM3S315，LM3S316，LM3S317，LM3S328，LM3S600，LM3S601，LM3S608，LM3S610，LM3S611，LM3S612，LM3S613，LM3S615，LM3S617，LM3S618，LM3S628，LM3S800，LM3S801，LM3S808，LM3S811，LM3S812，LM3S815， LM3S817，LM3S818，LM3S828，LM3S1110，LM3S1133，LM3S1138，LM3S1150，LM3S1162，LM3S1165， LM3S1332，LM3S1435，LM3S1439，LM3S1512，LM3S1538，LM3S1601，LM3S1607，LM3S1608，LM3S1620，LM3S1625，LM3S1626，LM3S1627，LM3S1635，LM3S1637，LM3S1751，LM3S1776，LM3S1850，LM3S1911，LM3S1918，LM3S1937，LM3S1958，LM3S1960，LM3S1968，LM3S2016，LM3S2110，LM3S2139，LM3S2276，LM3S2410，LM3S2412，LM3S2432，LM3S2533，LM3S2601，LM3S2608，LM3S2616，LM3S2620，LM3S2637，LM3S2651，LM3S2671，LM3S2678，LM3S2730，LM3S2739，LM3S2776，LM3S2911，LM3S2918，LM3S2939，LM3S2948，LM3S2950，LM3S2965，LM3S3739，LM3S3748，LM3S3749，LM3S3651，LM3S3759，LM3S3768，LM3S5632，LM3S5732，LM3S5737，LM3S5739，LM3S5747，LM3S5749，LM3S5652，LM3S5662，LM3S5752，LM3S5757，LM3S5762，LM3S5767，LM3S5768，LM3S5769，LM3S6100，LM3S6110，LM3S6420，LM3S6422，LM3S6432，LM3S6537，LM3S6610，LM3S6611，LM3S6618，LM3S6633，LM3S6637，LM3S6730，LM3S6753，LM3S6816，LM3S6911，LM3S6916，

LM3S6918，LM3S6938，LM3S6950，LM3S6952，LM3S6965，LM3S8530，LM3S8538，LM3S8630，LM3S8730，LM3S8733，LM3S8738，LM3S8930，LM3S8933，LM3S8938，LM3S8962，LM3S8970，LM3S8971，LM3S9790，LM3S9792，LM3S9997，LM3S9B90，LM3S9B92，LM3S9B95，LM3S9B96，LM3S9L97

其它型号芯片陆续支持

- **MARVELL（Intel）**

8051 内核 8031AH，8032AH，8051AH，8052AH，80C151SB，80C152JA，80C152JB，80C152JC，80C152JD，80C31BH，80C32，80C51BH，80C51FA，80C51GB，80C51RA，80C51SL_AH，80C51SL_AL，80C52，80C54，80C58，80L32，80L51FA，80L52，80L54，80L58，81C51SL_AH，81C51SL_AL，83C151SA，83C151SB，83C152JA，83C152JB，83C152JC，83C152JD，83C51FA，83C51FB，83C51FC，83C51GB，83C51RA，83C51RB，83C51RC，83C51SL_AH，83C51SL_AL，83L51FA，83L51FB，83L51FC，87C151SA，87C151SB，87C51BH，87C51FA，873L51FB，87C51FC，87C51GB，87C51RA，87C51RB，87C51RC，87C51SL_AH，87C51SL_AL，87C52，87C54，87C58，87L51FA，873L51FB，87L51FC，87L52，87L54，87L58

其它型号芯片陆续支持

XSCALE 内核

PXA255，PXA270

其它型号芯片陆续支持

- **Megawin**

8051 内核 MPC89E515A，MPC89E51A，MPC89E52A，MPC89E53A，MPC89E54A，MPC89E58A，MPC89L515A，MPC89L516X2，MPC89L51A，MPC89L52A，MPC89L53A，MPC89L54A，MPC89L556X2，MPC89L58A

其它型号芯片陆续支持

- **NUVOTON（WinBOND）**

8051 内核 W77C32/W77C032，W77E58/W77E058，W77E516，W77E532，W77LE58/W77L058，W77LE516/W77L516，W77LE532/W77L532，W78C32/W78C032，W78C51/W78C051，W78C521/W78C052，W78C54/W78C054，W78C801，W78E51/W78E051，W78E52/W78E052，W78E54/W78E054，W78E58/W78E058，W78E365，W78E516，W78E858，W78ERD2，W78IRD2，W78L32，W78L51，W78L52，W78L54，W78L801，W78LE51/W78L051，W78LE52/W78L052，W78LE54/W78L054，W78LE58/W78L058，W78LE516/W78L516，W78LE812/L812，W79E201，W79E532，W79E548，W79E549，W79E632，W79E633，W79E648，W79E649，W79L532，W79L548，W79L549，W79L632，W79L633，W79L648，W79L649

其它型号芯片陆续支持

ARM 内核 NUC501

其它型号芯片陆续支持

Cortex-M0 内核

NUC100LE3AN，NUC100RE3AN，NUC100RD3AN，NUC100VE3AN，NUC100VD3AN，NUC100VD2AN，NUC100LD2AN，NUC100LD1AN，NUC100LC1AN，NUC100RD2AN，NUC100RD1AN，NUC100RC1AN，NUC120LE3AN，NUC120LD3AN，NUC120RE3AN，NUC120RD3AN，NUC120VE3AN，NUC120VD3AN，NUC120VD2AN，NUC120LD2AN，NUC120LD1AN，NUC120LC1AN，NUC120RD2AN，NUC120RD1AN，NUC120RC1AN，NUC130LE3AN，NUC130LD3AN，NUC130LD2AN，NUC130RE3AN，NUC130RD3AN，NUC130RD2AN，NUC130VE3AN，NUC130VD3AN，NUC130VD2AN，NUC140LE3AN，NUC140LD3AN，NUC140LD2AN，NUC140RE3AN，NUC140RD3AN，NUC140RD2AN，NUC140VE3AN，NUC140VD3AN，NUC140VD2AN

其它型号芯片陆续支持

Flash 器件 W19B320AB，W19B320AT，W19B320BB，W19B320BT
其它型号 Flash 器件陆续支持

- **NXP**

8051 内核 P80C31，P80C31X2，P80C32，P80C32X2，P80C51FA，P80C51RA+，P80C552，P80C554，P80C557E4，P80C557E6，P80C557E8，P80C591，P80C592，P80CE598，P80C654X2，P80C660X2，P80C661X2，P87C51FA，P87C51FB，P87C51FB，P87C51MB2， P87C51MC2，P87C51MB2-02，P87C51MC2-02，P87C51RA+，P87C51RB+，P87C51RC+，P87C51RD+，P87C51RA2，P87C51RB2，P87C51RC2，P87C51RD2，P87C51，P87C51X2，P87C52，P87C52X2，P87C54，P87C54X2，P87C58X2，P87C552，P87C554，P87C557E6，P87C557E8，P87C591，P87C592，P87CE598，P87C654X2，P87C660X2，P87C661X2，P89C51，P89C51X2，P89C52，P89C52X2，P89C54，P89C54X2，P89C58，P89C58X2，P89C51RA2，P89C51RB2，P89C51RC2，P89C51RD2，P89C51RA2H，P89C51RB2H，P89C51RC2H，89C51RD2H，P89C51RC2+， P89C51RD2+，P89LV51RD2，P89C60X2，P89C61X2，P89C557E4，P89C660，P89C662，P89C664，P89C668，P89C669，P89V52X2，P89V660，P89V662，P89V664
其它型号芯片陆续支持

ARM 内核 LPC2101，LPC2102，LPC2103，LPC2104，LPC2105，LPC2106，LPC2109，LPC2114，LPC2119，LPC2124，LPC2129，LPC2194，LPC2131，LPC2132，LPC2134，LPC2136，LPC2138，LPC2141，LPC2142，LPC2144，LPC2146，LPC2148，LPC2157，LPC2158，LPC2194，LPC2210，LPC2212，LPC2214，LPC2220，LPC2290，LPC2292，LPC2294，LPC2364，LPC2365，LPC2366，LPC2367，LPC2368，LPC2377，LPC2378，LPC2387，LPC2388，LPC2458，LPC2460，LPC2468，LPC2470，LPC2478，LPC2880，LPC2888，LPC2917，LPC2919，LPC3130，LPC3131，LPC3141，LPC3143，LPC3152，LPC3154，LPC3180，LPC3220，LPC3230，LPC3240，LPC3250，LH7A400，LH7A404
其它型号芯片陆续支持

Cortex-M3 内核
LPC1751，LPC1752，LPC1754，LPC1756，LPC1758，LPC1764，LPC1765，LPC1766，LPC1768，LPC1311，LPC1313，LPC1342，LPC1343
其它型号芯片陆续支持

Cortex-M0 内核
LPC1111x101，LPC1111x201，LPC1112x101，LPC1112x201，LPC1113x101，LPC1113x201，LPC1114x101，LPC1114x201
其它型号芯片陆续支持

- **OKI**

8051 内核 MSM80C154S，MSM80C31F，MSM80C51F，MSM83C154S
其它型号芯片陆续支持

ARM 内核 全系列 ARM 内核，即将支持

- **Samsung**

ARM 内核 S3C44B0X，S3C2410A，S3C2416X，S3C2440A，S3C4510，S3C6410
其它型号芯片陆续支持

- **Silicon Labs**

8051 内核 **C8051F00x/01x 系列**
C8051F000，C8051F001，C8051F002，C8051F005，C8051F006，C8051F007，C8051F010，C8051F011，C8051F012，C8051F015，C8051F016，C8051F017
其它型号芯片陆续支持

C8051F02x 系列

C8051F020，C8051F021，C8051F022，C8051F023

其它型号芯片陆续支持

C8051F04x 系列

C8051F040，C8051F041，C8051F042，C8051F043，C8051F044，C8051F045，C8051F046，C8051F047

其它型号芯片陆续支持

C8051F06x 系列

C8051F060，C8051F061，C8051F062，C8051F063，C8051F064，C8051F065，C8051F066，C8051F067

其它型号芯片陆续支持

C8051F2xx 系列

C8051F206，C8051F220，C8051F221，C8051F226，C8051F230，C8051F231，C8051F236

其它型号芯片陆续支持

C8051F30x 系列

C8051F300，C8051F301，C8051F302，C8051F303，C8051F304，C8051F305

其它型号芯片陆续支持

C8051F31x 系列

C8051F310，C8051F311，C8051F312，C8051F313，C8051F314，C8051F315，C8051F316，C8051F317

其它型号芯片陆续支持

C8051F32x 系列

C8051F320，C8051F321，C8051F326，C8051F327

其它型号芯片陆续支持

C8051F33x 系列

C8051F330，C8051F331，C8051F332，C8051F333，C8051F334，C8051F335，C8051F336，C8051F337，C8051F338，C8051F339

其它型号芯片陆续支持

C8051F34x 系列

C8051F340，C8051F341，C8051F342，C8051F343，C8051F344，C8051F345，C8051F346，C8051F347，C8051F348，C8051F349，C8051F34A，C8051F34B

其它型号芯片陆续支持

C8051F35x 系列

C8051F350，C8051F351，C8051F352，C8051F353

其它型号芯片陆续支持

C8051F36x 系列

C8051F360，C8051F361，C8051F362，C8051F363，C8051F364，C8051F365，C8051F366，C8051F367，C8051F368，C8051F369

其它型号芯片陆续支持

C8051F41x 系列

C8051F410，C8051F411，C8051F412，C8051F413

其它型号芯片陆续支持

C8051F50x/51x 系列

C8051F500，C8051F501，C8051F502，C8051F503，C8051F504，C8051F505，C8051F506，C8051F507，C8051F508，C8051F509，C8051F510，C8051F511

其它型号芯片陆续支持

C8051F52x/53x 系列

C8051F520，C8051F521，C8051F523，C8051F524，C8051F526，C8051F527，C8051F530，C8051F531，C8051F533，C8051F534，C8051F536，C8051F537

其它型号芯片陆续支持

C8051F70x/71x 系列

C8051F700，C8051F701，C8051F702，C8051F703，C8051F704，C8051F705，C8051F706，C8051F707，C8051F708，C8051F709，C8051F710，C8051F711，C8051F712，C8051F713，C8051F714，C8051F715

其它型号芯片陆续支持

C8051F92x/93x 系列

C8051F920，C8051F921，C8051F930，C8051F931

其它型号芯片陆续支持

C8051F12x/13x 系列

C8051F120，C8051F121，C8051F122，C8051F123，C8051F124，C8051F125，C8051F126，C8051F127，C8051F130，C8051F131，C8051F132，C8051F133

其它型号芯片陆续支持

C8051F58x/59x 系列

C8051F580，C8051F581，C8051F582，C8051F583，C8051F584，C8051F585，C8051F586，C8051F587，C8051F588，C8051F589，C8051F590，C8051F591

其它型号芯片陆续支持

- **SST**

8051 内核 SST89E516RD，SST89E516RD2，SST89E51RC，SST89E52RC，SST89E52RD，SST89E52RD2，SST89E54RC，SST89E54RD，SST89E54RD2，SST89E54RD2A，SST89E54RDA，SST89E58RD，SST89E58RD2，SST89E58RD2A，SST89E58RDA，SST89E554RC，SST89E564RD，SST89V516RD，SST89V516RD2，SST89V51RC，SST89V52RD，SST89V52RD2，SST89V54RD，SST89V54RD2，SST89V54RDA，SST89V54RD2A，SST89V58RD，SST89V58RD2，SST89V58RDA ，SST89V58RD2A，SST89V554RC，SST89V564RD

其它型号芯片陆续支持

Flash 器件 SST36VF1601，SST36VF1602，SST36VF3203，SST36VF3204，SST39LF200A，SST39LF400A，SST39LF800A，SST39VF1601，SST39VF1602，SST39VF3201，SST39VF3202，SST39VF6401，SST39VF6402，SST39WF400A，SST39WF800A，SST39WF1601，SST39WF1602

其它型号 Flash 器件陆续支持

- **ST**

ARM 内核 STR710FZ1，STR710FZ2，STR711FR0，STR711FR1，STR711FR2，STR712FR0，STR712FR1，STR712FR2，STR715FR0，STR730FZ1，STR730FZ2，STR731FV0，STR731FV1，STR731FV2，STR735FZ1，STR735FZ2，STR736FV0，STR736FV1，STR736FV2，STR750FV0，STR750FV1，STR750FV2，STR751FR0，STR751FR1，STR751FR2，STR752FR0，STR752FR1，STR752FR2，STR755FR0，STR755FR1，STR755FR2，STR755FV0，STR755FV1，STR755FV2，STR910FM32X6，STR910FW32X6，STR910FAZ32H6，STR911FM42X6，STR911FM44X6，STR912FW42X6，STR912FW44X6，STR912FAZ42H6

其它型号芯片陆续支持

Cortex-M3 内核

STM32F101C4，STM32F101R4，STM32F101T4，STM32F101C6，STM32F101R6，STM32F101T6，STM32F101C8，STM32F101R8，STM32F101V8，STM32F101T8，STM32F101RB，STM32F101VB，STM32F101CB，STM32F101RC，STM32F101VC，STM32F101ZC，STM32F101RD，STM32F101VD，STM32F101ZD，STM32F101RE，STM32F101VE，STM32F101ZE，STM32F102C4，STM32F102R4，STM32F102C6，STM32F102R6，STM32F102C8，STM32F102R8，STM32F102CB，STM32F102RB，STM32F103C4，STM32F103R4，STM32F103T4，STM32F103C6，STM32F103R6，STM32F103T6，STM32F103C8，STM32F103R8，STM32F103V8，STM32F103T8，STM32F103RB，STM32F103VB，STM32F103CB，STM32F103RC，STM32F103VC，STM32F103ZC，STM32F103RD，STM32F103VD，STM32F103ZD，STM32F103RE，STM32F103VE，STM32F103ZE

其它型号芯片陆续支持

STC

8051 内核 STC89C51RC，STC89C52RC，STC89C53RC，STC89C54RD+，STC89C55RD+，STC89C58RD+，STC89C516RD+，STC89LE51AD，STC89LE52AD，STC89LE54AD，STC89LE58AD，STC89LE516AD，STC89LE51RC，STC89LE52RC，STC89LE53RC，STC89LE54RD+，STC89LE58RD+，STC89LE516RD+，STC89LE516X2

其它型号芯片陆续支持

SyncMOS

8051 内核 SM5964AL，SM5964C，SM59128C，SM59264，SM7964AL，SM7964C， SM79108C，SM79108L，SM79164C，SM79164L，SM79164V，SM80C51C，SM80C51L，SM80C52C，SM80C52L，SM80C58C，SM80C58L，SM8951AC，SM8951AL，SM8951BC，SM8951BL，SM8952AC，SM8952AL，SM8954AC，SM8954AL，SM8958AC，SM8958AL，SM89516AC，SM89516AL，SM89516C，SM89516L，SM894051C，SM894051L，SM89S08R1C，SM89S08R1L，SM89S16R1C，SM89S16R1L，SM89T08R1C，SM89T08R1L

其它型号芯片陆续支持

TI

8051 内核 MSC1200Y2，MSC1200Y3，MSC1201Y2，MSC1201Y3，MSC1202Y2，MSC1202Y3，MSC1210Y2，MSC1210Y3，MSC1210Y4，MSC1210Y5，MSC1211Y2，MSC1211Y3，MSC1211Y4，MSC1211Y5，MSC1212Y2，MSC1212Y3，MSC1212Y4，MSC1212Y5，MSC1213Y2，MSC1213Y3，MSC1213Y4，MSC1213Y5，MSC1214Y2，MSC1214Y3，MSC1214Y4，MSC1214Y5

其它型号芯片陆续支持

ARM 内核 TMS470R1A64，TMS470R1A128，TMS470R1A256

其它型号芯片陆续支持

DSP 内核 **DaVinci™ 数字媒体处理器**

TMS320DM6467-594，TMS320DM6467-729，TMS320DM6446-594，TMS320DM6446-513，TMS320DM6443-594，TMS320DM6441-513，TMS320DM6441-405，TMS320DM6437-700，TMS320DM6437-600，TMS320DM6437-500，TMS320DM6437-400，TMS320DM6435-700，TMS320DM6435-600，TMS320DM6435-500，TMS320DM6435-400，TMS320DM6433-700，TMS320DM6433-600，TMS320DM6433-500，TMS320DM6433-400，TMS320DM6431-300，TMS320DM648-900，TMS320DM648-720，TMS320DM647-900，TMS320DM647-720，TMS320DM642-720，TMS320DM642-600，TMS320DM642-500，TMS320DM641-600，TMS320DM641-500，TMS320DM640-400，TMS320DM357，TMS320DM355-270，TMS320DM355-216，TMS320DM355-135，TMS320DM335-216，TMS320DM335-135

OMAP™ 应用处理器

OMAP3530，OMAP3525，OMAP3515，OMAP3503，OMAP-L137

C2000™ 系列 32 位实时 MCU

TMS320F28335，TMS320F28334，TMS320F28332，TMS320F28035，TMS320F28027，TMS320F28023，TMS320C2801，TMS320C2802，TMS320C2810，TMS320C2811，TMS320C2812，TMS320F2801-100，TMS320F2801-60，TMS320F28015，TMS320F28016，TMS320F2802-100，TMS320LF2407A，TMS320LF2406A，TMS320LF2403A，TMS320LF2402A，TMS320LF2401A，TMS320LC2406A，TMS320LC2404A，TMS320LC2403A，TMS320LC2402A，TMS320LC2401A

C5000™ 低功耗 DSP

TMS320VC5510A-200，TMS320VC5510A-160，TMS320VC5509A-200，TMS320VC5507-200，TMS320VC5506-108，TMS320VC5503-200，TMS320VC5502-300，TMS320VC5502-200，TMS320VC5501-300，TMS320VC549-120，TMS320VC549-100，TMS320VC5471，TMS320VC5470，TMS320VC5441-532，TMS320VC5421-200，TMS320VC5420-200，TMS320VC5416-160，TMS320VC5416-120，TMS320VC5410A-160，TMS320VC5410A-120，TMS320VC5410-100，TMS320VC5409A-160，TMS320VC5409A-120，TMS320VC5409-80，TMS320VC5409-100，TMS320VC5407-120，TMS320VC5404-120，TMS320VC5402A-160，TMS320VC5402-100

C6000™ DSP 平台

TMS320C6474，TMS320C6455-850，TMS320C6455-720，TMS320C6455-1200，TMS320C6455-1000，TMS320C6454-850，TMS320C6454-720，TMS320C6454-1000，TMS320C6452-900，TMS320C6452-720，TMS320C6416T-850，TMS320C6416T-720，TMS320C6416T-600，TMS320C6416T-1000，TMS320C6416-7E3，TMS320C6416-6E3，TMS320C6416-5E0，TMS320C6415T-850，TMS320C6415T-720，TMS320C6415T-600，TMS320C6415T-1000，TMS320C6415-7E3，TMS320C6415-6E3，TMS320C6415-5E0，TMS320C6414T-850，TMS320C6414T-720，TMS320C6414T-600，TMS320C6414T-1000，TMS320C6414-7E3，TMS320C6414-6E3

C6000™ 性能值 DSP

TMS320C6424-700，TMS320C6424-600，TMS320C6424-500，TMS320C6424-400，TMS320C6421-700，TMS320C6421-600，TMS320C6421-500，TMS320C6421-400，TMS320C6418-600，TMS320C6418-500，TMS320C6413-500，TMS320C6412-720，TMS320C6412-600，TMS320C6412-500，TMS320C6411-300，TMS320C6410-400，TMS320C6211B-167，TMS320C6211B-150，TMS320C6205-200，TMS320C6204-200，TMS320C6203B-300，TMS320C6203B-250，TMS320C6202B-300，TMS320C6202B-250，TMS320C6201-200

C6000™ 浮点 DSP

TMS320C6745-200，TMS320C6745-300，TMS320C6747-200，TMS320C6747-300，TMS320C6727B-350，TMS320C6727B-300，TMS320C6727B-275，TMS320C6727B-250，TMS320C6726B-266，TMS320C6726B-225，TMS320C6722B-250，TMS320C6722B-225，TMS320C6722B-200，TMS320C6720-200，TMS320C6713B-300，TMS320C6713B-225，TMS320C6713B-200，TMS320C6713B-167，TMS320C6712D-150，TMS320C6711D-250，TMS320C6711D-200，TMS320C6711D-167，TMS320C6701-167，TMS320C6701-150，SM320C6713B-EP

其它型号芯片陆续支持

- **TOSHIBA**

Cortex-M3 内核

TMPM330FDFG，TMPM330FYFG，TMPM330FWFG，TMPM332FWUG，TMPM370FYDFG，TMPM370FYFG，TMPM380FYDFG，TMPM380FYFG

其它型号芯片陆续支持